A Preliminary Study of Recommender System Algorithms for MathOverflow Users

Yifeng Song, 03/05/2017

Introduction

MathOverflow (http://mathoverflow.net/) is a forum where registered users can ask and answer research level mathematics questions, which is a part of the StackExchange website where the discussion forums for various disciplines are hosted (StackOverflow is their most famous subsection; it is created for software developers to discuss and share their problems). It can be noticed that there are lots of questions which have been posted online for a quite long time but still did not receive any answers, so if a recommender system can automatically match questions to registered users who are potentially interested in them and are able to give correct answers, not only the users who asked the questions but also the broader public who can have access to the forum will benefit from the answered questions. Also, a forum with increased user activity will potentially help boost the growth and improvement of the website itself. Using the historical questions/answers & users data collected by StackExchange during 2009 – 2016, I tried to build a recommender system which provides personalized recommendations to each user with the questions that they are likely to be interested in and be able to answer. The recommender system is basically a predictive model which calculates the scores of all active items (in this study, questions that were asked) in the system for each user, and recommends the items with the highest scores to the user. Ideally, the system could match the items (resources) which best matches the user. The list of items will get different rankings for different users, based on the user preference/interests as well as the item properties.

Data

The data are available for downloading from the StackExchange website (https://archive.org/download/stackexchange). The raw data are in the XML format, which contains the information for each question and answer, including the asker's id, the question/answer's title and text body, the tags associated with the questions, which question the answer corresponds to, the answerer's id, the time when the question/answer was posted, etc.. Also included in the data are the user's basic information. The data were cleaned and reorganized using Python scripts. The entire data set contains 78,048 questions related to mathematics, and the total number of users who have answered at least one question is 11,166. There are a lot more registered users on the site, but since most of them did not ever answer a question, those people are not very interesting in the sense of recommending questions to users. Moreover, the majority of the 11,166 users only answered a question once before, so those users are excluded from the data used in the modelling because if the user only answered once, when we predict which questions this user will be mostly likely to answer in the future, there is no way of evaluating the model's prediction accuracy on this particular user. There are 4,513 users left after the users who answered only one question were excluded. For this user group, they have provided a total number of 105,796 answers to the question pool. If we define a user-item interaction as one user answers a question, the interactions can be denoted by a matrix, with each row representing a user, and each column representing a question. And the matrix element at (i,j) represents whether the user i answered the question j. If so, the value at this position is 1, and otherwise it will be a missing value. Each interaction has a timestamp which stands for the date and time when the answer to the question j was posted by user i. To conduct the predictive modelling, the interactions data are split into training and testing part – the first 90% of the interactions are placed in the training set (assuming those data are all historical data we have), and the most recent 10% of the interactions are placed in the testing set. The model will be built on the training set, and assuming the latest 10% interactions in the testing set are in the future, the model can then be used to recommend the questions to the users and see if the model is able to give high rankings to the questions which were actually answered by them during the testing set period. In this way, the predictive power of the model can be evaluated and the performances of the different modelling approaches can be compared against each other.

Approaches & Procedures

1) Principle of recommender system approaches

The most important question is how we could find out the questions that best matches each individual given such a large pool of users and questions. Basically, there are two types of methods of building recommender systems. The first one is called **content-based filtering recommender system**. The intuition behind it is that the questions might be about different topics and the users have different tastes or interests for different topics, so based on historical interaction data we could build models to learn to which degree each user is interested in different topics. Then for a question which has not been answered by a certain user, as long as we know its contents, we will know if the contents of this question is similar to the contents of the questions that have been answered by the user, then it will be easy to decide whether this question will be suitable to him/her. In addition, we can not only leverage the content information of the items (questions being asked), but also use the user's 'content' information, such as the demographics. For instance, we could categorize users into different groups, with each group similar demographic features. Then the content-based filtering algorithm could try to learn for each individual item, which group will be most interested in it. The second type of method is called **collaborative filtering recommender system**. The intuition behind this algorithm is that if most questions that were answered by user A have also been answered by user B, then these two users might be very similar to each other in terms of their interests in questions, so for a question that is answered by user B but not by user A, if we try to recommend this question to user A, then there is a decent chance that A will be interested in the question. While if user C does not answered many common questions with user A, then recommending the questions answered by user C to user A will not have a good chance of success. Likewise, for two questions x and y, if they share very similar group of users who answered them, they should have a lot in common in their nature. So if a user answered question x but did not answer question y, we might try to recommend y to him as well assuming question x and y are highly correlated. In short, content-based filtering focuses on understanding the static properties of each individual user and question in the system, while collaborative filtering is based on learning the interrelated and dynamic user behavior. The two approaches can give distinct results for the same data – for instance, collaborative model might recommend a question to a user because the users similar to this user all answered that question, but content-based model predicts that the question is not suitable for this user based on the contents of the questions answered by this user. Unless we have sufficient amount of data for validation, we will not be able to tell which model is correct in each particular user case. But the two methods could be combined to complement each other to achieve improved accuracy as a **hybrid recommender system**, if it is done appropriately. In fact, if we assume that the way users answer questions on MathOverflow website does not happen by chance, the interactions data might tell us additional information about the properties of both the questions and users, which is not reflected in the explicit properties/features of the questions and users. As a concrete example, let us say there are two questions whose topics and words are not quite similar to each other, but they are answered by similar group of users. This is where collaborative filtering comes into play – the two questions should share some common hidden (latent) factors, although they are not clear in the texts of the questions (it may not be hard for a human being to identify such hidden factors by inferring from the texts, but for a computer it is almost impossible). It might be that the two questions are concerned with different problems, but in reality they could solved by the same mathematical theory or principle. Or it could be anything else we do not know. Hybrid recommender system allows for setting those unknown latent factors as parameters, which could be estimated by fitting the training data.

2) Construction of feature set for questions & users

To build the hybrid recommender system model, we not only need the user-question interactions data split into training and testing part (as discussed in previous section), but also the features of both questions and users. As for questions, there are roughly 3 – 5 tags associated with each question (which are supposed to improve the search-by-relevance effectiveness of the question), which provides valuable information about what topics or subject areas each question is about. Each tag could be considered as a binary feature for each question, with 1 standing for the tag being associated with the question, and 0 standing for the tag not being associated with the question. Other potentially useful features could be extracted from the title and text body of the questions. The tags could help categorize the questions on a high level, but by digging into the texts of questions, more detailed information about each particular question might be found. However, working with texts is quite challenging, as the raw texts must be converted into numerical formats in order to be consistent with the quantitative modeling techniques. Lots of assumptions and simplifications have to be made in the process of cleaning and transforming the raw text data, and thus part of the information carried with the texts will be lost. One popular method of representing texts is called document-term matrix. By treating each question (together with its title) as a 'document', the counts of each distinct word contained in each document are generated and combined to form a single matrix, in which the element at position (d,w) is the count of word w in document d. But since there are a huge number of distinct words present in the all ~ 80,000 questions, it is not desirable to directly feed the document-term matrix into the model as the input item features, otherwise there will be too many model parameters which will make the model very prone to overfitting (the model fits the training data really well and the prediction is poor when the model is generalized to the testing data unseen during the training step). Therefore topic modeling, a text mining method is used to solve this issue through dimensionality reduction. It uses a probabilistic machine learning algorithm (named Latent Dirichlet Allocation) to automatically identify topics present in a text object, that is, a repeating pattern of co-occurring terms in a document. A topic is composed of many words with specific proportions, and each document (question) can be about just one topic or a mixture of several topics with different weights. We have a large set of questions which include huge number of distinct words, but in fact there are much less different topics than questions as many questions are very similar to each other. Also, when recommending questions to users, usually it is what the question is actually talking about rather than the presence or counts of certain individual words (without their contexts) that determine whether the question is a good match with the user. Therefore, the topical compositions of each document resulting from topic modeling could be used the item features in addition to the questions tags, which has the advantage of reduced dimensionality/complexity and more meaningful information extraction from the texts. To perform topic modeling, the raw texts of the questions need to first go through a Natural Language Processing (NLP) pipeline that standardizes the data such that the HTML tags, punctuation and stop words (words that are common but do not carry a real meaning, like this, a, from, around, etc.) are removed and the text strings are split into individual words. Then document-term matrix could be created and serve as the starting point for topic modeling. Finally, the topical compositions of each document could be extracted as the model output and concatenated with the item feature vector for question tags, completing the construction of the item feature set. The following is an example of some of the topics of the question pool generated from topic modeling (the top 10 words of each topic are listed):

Topic 2: homotopy, equivalence, relation, isomorphism, diagram, transformation, loop, equivalent, preserve, bijection

Topic 6: subset, measure, continuous, measurable, borel, dense, xin, closed, topological, lebesgue

Topic 24: graph, vertex, tree, edge, node, cycle, spanning, rooted, connected, planar

Topic 26: random, probability, distribution, variable, density, interval, independent, cardinal, size, Gaussian

Topic 42: matrix, eigenvalue, vector, symmetric, dot, linear, entry, column, tau, diagonal

Based on the word combinations of each topic, topic 2 should be about group theory in abstract algebra, topic 6 is likely to be about the Lebesgue integral & measure theory in real analysis, topic 24 is probably about the tree structure in graph theory, topic 26 should be about the random variable and probability distribution in probability theory, and topic 42 is probably about matrix diagonalization in linear algebra.

But there are topics of which the meaning is not very clear:

Topic 1: orbit, mathfrak, operatornamead, oplus, nilpotent, expectation, subalgebra, cartan, conditional, ast

Topic 20: subject, understand, kappa, mean, concentration, research, related, cardinality, bit, analysis

One issue with those topics is that they contain some 'words' which are not actually English words (the questions sometimes mathematical notations and equations which contain strange 'words'), or the words which are not related to the specific topics of mathematics (such as understand, research, related). So this demonstrated the challenge of text mining – in order to extract high quality features from the raw texts, lots of efforts need to be put into the data preprocessing and modeling step to find the best practice. But for this preliminary case study, although the topical features extracted are not perfect, they should at least provide some insights into the recommendation algorithm.

As for the user features, the data provided by StackExchange does not provide much relevant user profile information, except the 'About Me', which is user's brief self-introduction that might reveal the user's mathematical background or interests. So the collection of all 'About Me' texts were trained by topic modeling to generate user features. Also, every single answer that has been ever provided by each user could be extracted, so for each user his/her 'document' will be the texts of all answers he provided before. Then we could apply topic modeling to look for features based on their answers, assuming those features could help find the best matching questions to them. Finally, the features from 'About Me' and features from user's answers are merged, completing the construction of user feature set.

3) Methods of training the machine learning algorithm for hybrid recommender systems

There are different approaches to model the collaborative filtering part of the hybrid recommender system. In this study, the latent factor representation is used where the k latent factors of each question and k latent factors of each user are both represented as k-dimensional vectors (k is a hyper-parameter of the model, which could be tuned to achieve improved prediction accuracy during the model evaluation phase). The values within each latent factor vector are unknown parameters of the model, which will be estimated after fitting the model to training data. For the content-based part of the hybrid recommender system, there will be vectors of weight parameters corresponding to the features of each question (tags

+ question topics) as well as vectors of weight parameters corresponding to the features of each user ('AboutMe' + answer topics), both of which are unknown parameters to be estimated in fitting the data. To calculate the score of question q to user u, we need to calculate the dot product of latent factor vector for user u and latent factor vector for question q, the dot product of item feature vector of question q and its corresponding weight vector for user u, the dot product of user feature vector of user u and its corresponding weight vector for question q, and add them together. Finally a transformation function is applied to the value to return the final score. Since the predicted score for each user-question pair could be calculated if all model parameters are known, the question list could be ranked for each user based on the predicted score with the questions receiving the highest scores getting recommended to the user. Thus the goal of the model is to maximize the average predicted ranking of the questions that have been actually answered by each user in the training set. Generally speaking, for a machine learning problem, in order to maximize the consistency between the predicted target values and the real target values, we could define an objective function called "loss function" which provides an estimation of the overall prediction errors on the training data. Then the machine learning problem can be converted into a numerical optimization problem, in which the goal is to find out a set of parameters which could minimize the loss function. The user-question interactions data we have belong to the implicit feedback type – that is, we only have positive observations (in this case, if a user answered a question, it is a positive observation; but if he/she does not answer a question, we do not know if the user will not answer it, as it is likely that the user never got the chance to see the question) which makes it hard to define the target value for prediction and the loss function. This is in contrast to the explicit feedback type (like the product ratings on Netflix, Yelp, etc.) where the ratings are on a numerical scale, so a low rating clearly indicates that the user does not like the item, and the target values can be simply the rating. Fortunately there are methods being proposed to deal with the loss function for implicit feedback recommender systems in recent years which have been proved to have good performance. In this work, I chose WARP loss function (WARP is the abbreviation of weighted approximate-rank pairwise), which applies a negative sampling technique to estimate the deviation of predicted ranking of the positive observations from actual ranking of the positive observations using an approximation function whose functional form allows for finding its minima through efficient numerical optimization. The recommender system model in this case study is trained using LightFM, a Python library that offers the framework for implementing and testing several recommendation algorithms.

Results Evaluation & Discussion

After model fitting part is done, we obtained the estimated parameters of the model. Those parameter values could then be used to predict all recommendations in the testing set, giving a list of questions ranked by their predicted scores for each user (except for the questions which were already answered by this user during the period of training set: they will not be re-recommended to the user). In practice, the recommender system will predict a fixed number of questions to each user, and the model performance could be evaluated by calculating what percentage of the questions that were actually answered by the user in the testing set end up being recommended to him/her according to model prediction. This metric is defined as recall at k (k stands for the number of questions recommended to the user). The higher the recall value is, the better the recommender system's ability of correctly identifying the fitting questions to the user is. However, there are also issues with this metric. For instance, if a user only answered one question during the period of the testing set, and the system recommends 20 questions to him/her, one of which is the one he actually answered. The recall at k (k=20) for this user is 100%, which is a perfect score, but we do not know if the rest of the 19 questions are also good fits to this user. Therefore we need another metric to deal with this problem – precision at k. It is defined as follows: among all questions recommended to the user, what percentage of them are actually answered by the user. So in this example, although the recall at k is 100% for this user, the precision at k is just 5%. There are tradeoffs between recall and precision: in one extreme case, we could recommend all questions to the user, which apparently makes no sense in real applications, we will get a perfect recall value, but the precision is almost zero; in the other extreme case, we could recommend only one question to the user and if we are lucky enough to pick up the question that is answered by the user, the precision is 100%, but if the user actually answered many questions, we will not think it is a good recommendation practice since the recall value is low. As a result, it is not a good idea to compare the performance of two recommendation models using either precision or recall at fixed k value, because one model might outperform the other at a certain k value in terms of recall or precision, but if k takes on a different value, the change of k might drastically change the recall and precision of the prediction, making the model inferior to the other. Besides, selecting an appropriate k value (number of questions recommended to each user) in practice is very tricky, since a too small k value will fail to recommend sufficient number of questions to interested users, while a too big k value has the risk of recommend more irrelevant questions to users or leads to information overloading. So in this case study, I did not try to specify a k value to calculate the model's recall and precision, but used another evaluation metric instead for simplicity and avoiding the confusion of precision-recall tradeoff. The metric is called AUC (abbreviation for area under curve). To calculate AUC score of the recommendation model for each user, we calculate the true positive rate (TPR: same as recall) and false positive rate (FPR: among all questions that were not answered by the user, what percentage of those questions were recommended to the user by the model) at each different k value. For example, we could start from k=0, where no questions are recommended to the user, in this case both TPR and FPR are zero. Then we calculate the values of TPR and FPR at k=1, and then k=2, and so on. Finally, when k is equal to the total number of questions, both TPR and FPR are equal to 1 because all questions regardless of whether the question is answered by the user or not are recommended. When we plot the set of TPR values calculated at different k values versus the set of FPR values calculated at each corresponding k value, we obtain the ROC (receiver operating characteristic) curve, and the integration area under this curve is the AUC score. When comparing the predictive performance of two different recommender system algorithms, we could plot the ROC curves of the two models in the same graph. If model I's ROC curve is above model II's, in order for the two models to achieve same level of TPR (or recall, say, 80%), the corresponding FPR of model I will be smaller than model II, and a smaller FPR means model I recommends less number of 'negative' questions (questions which were actually not answered by him/her) to the user, and this is equivalent to a higher precision. Likewise, in order for the two models to achieve same level of FPR, model I's TPR is higher than model II, so the number of questions correctly recommended by model I is higher than model II, therefore both the recall and precision of model I is better than model II. In more complicated cases where model I's ROC curve is above model II in certain ranges but blow it in other ranges (two ROC curves cross with each other), AUC score could be used to find which model is superior (the higher, the better) in the sense of averaging over all possible k values. In this preliminary study we do not have prior knowledge about the best choice of k, so AUC will provide a reasonable evaluation of the overall performance of the recommendation model, which also has the advantage of taking both recall and precision metrics into account. AUC score falls between 0 and 1, with 1 being the perfect score in which case the model ranks every question that was actually answered by the user to the top of its recommendation list. An AUC score which is near or below 0.5 indicates that the model does not do a significantly better job than random recommendation. Finally, because each user has a different AUC score based on the model's personalized recommendation, the average AUC for all users are used for evaluation and determining which recommendation model works best.

I started with building a pure collaborative filtering recommender system by only using the user-question interactions data available in the training set. It can be considered as a special case of the hybrid recommender system – instead of using the extracted question and user features (described in previous section), just setting the feature vectors of all users and questions to be identical. The mean AUC score on the testing set is just 0.5178, which is barely better than random guessing. But this result is not surprising: considering the fact that there are 6,458 new questions showing up in the testing set; in other words, those questions were not available during the time period of the training set, but the majority of the answers (92.8%) provided by users during the time period of the testing set are answering those questions. Since collaborative filtering algorithm is trying to understand the correlations of users in terms of their interactions with the items as well as the correlations of items in terms of their interactions with the users, if the question has never been answered or it is an incoming new question, essentially no latent factors could be inferred for this item. Same principle applies to a new user entering the system or an old user who has never answered a question before. But in real-life recommender system applications it is important to recommend new questions to appropriate experienced users so that those questions will not die in the unnoticed status, and it is equally important to recommend questions to new users that they are likely to be interested in to get them up to speed in the system. By adding the explicit user and item features (features for content-based filtering) into the model, presumably the model will do a better job of recommending new questions to users by learning the question features that each individual user will be interested in and the user features that best match with each individual question from all historical user-question interactions data.

So the next step is to try hybrid recommender systems and evaluate their prediction performance on the testing set. First, the tag features of the questions are incorporated into the model. There are a total of 1,380 distinct tags associated with the 78,048 questions in the system, which are supposed to be the most reliable question features because they are explicitly provided by StackExchange without having to be cleaned or transformed beforehand, and the tags directly tell us about the topics, categories and contents of the question. The AUC score of this model is 0.8687, which is a significant improvement from the pure collaborative filtering model. This demonstrates that the inclusion of question features helps improve the recommendation accuracy of the model, especially for the system where there are often new questions being posted. To check if the topical features extracted from the raw texts of the questions could add additional value to the recommender system, the 50 topical features of the questions were merged with the 1,380 tag features to form a new item feature set, and a new hybrid model is built using the training set. The AUC score of this model is 0.8793, which is just a marginal increase from the previous model whose contents only include tag features of questions. Although there is not much improvement, the result still indicates that there is useful information stored in the texts of questions which are not contained in the tags of questions, and such information could help build better personalized recommendations. In other recommender systems we may not have the luxury of having access to the informative item features such as the question tags used in StackExchange, so we have to rely heavily on extracting the features & information from the unstructured data like the raw texts of the questions in this case study. Thus an interesting question is if we could still build an effective hybrid recommender if the tags data are not available to us. So the 4th model is tried, which only includes the 50 topical features of the questions as the contents of the recommender system. The AUC score of this model on the testing set is 0.8201, which is not comparable to the models that contain tags information in their feature set, but still has a significant advantage over the pure collaborative filtering model. This suggests that the topical features of the questions extracted using text mining techniques (topic modeling approaches) are highly overlapped with the tag features of the questions, although as an alternative to the tag features the quality of topical features is not equally good. The overlapping between the two types of features can also explain why there is only a marginal increase in AUC when the topical features are incorporated into the model in addition to the tag features.

After the item (question) features are fully explored, user features are added to the hybrid recommender system to test their usefulness. However, it turned out that instead of further improving the predictive model, they ended up lowering the AUC scores. When the topical features of the user's answers are added to the hybrid model where both the tag and topical features of the questions are included, the AUC score dropped to 0.8388. And the topical features extracted from each user's "AboutMe" are even worse: when those features are added to the hybrid model that contains the full item contents (tag + topical features), the AUC on the testing set is only 0.5517. Most likely, the cause of decreased AUC scores in those hybrid models that contain user features is overfitting. The topical features extracted from each user's answer collection do not seem to have a satisfactory quality, and this is probably because the length of the answer collection varies a lot by user (some users answer a lot of questions while others do not). Also, there are more special mathematical equations, operators and notations present in the texts of answers than in questions, which could potentially affect the robustness of topic modeling. In terms of the topical features extracted from each user's 'AboutMe' section, not only are there many users who left this part empty, but also user's self-introduction is oftentimes not relevant to mathematics. With those low quality and questionable user features being the input of the model, the machine learning algorithm still tries to fit the model so that the agreement between model predictions and the observed user-question interactions in the training set is as good as possible. But when the model is generalized to the unseen data (testing set), the presence of noisy features will cause the predictions to deviate from the true values. One way of alleviating this issue is to design more advanced natural language processing and text mining strategies to handle the user data so that more meaningful and less noisy features could be extracted. It is also possible that those user features are indeed not helpful for question recommendation, and the truly useful user features (such as user profile/demographic information) are not available in the data provided by StackExchange. Therefore, the worthlessness of the user features in this case study does not necessarily mean that the user features are not essential in other recommender systems.

In the MathOverflow question pool, there are 6,458 questions which were posted onto the system during the time period of testing set and have got at least one answer, and there are another 21,817 questions which have never been answered during the time periods of both training and testing sets. All of the models discussed above use all MathOverflow questions in the evaluation steps, which closely approximates the recommender systems used in real life, as we have to include new items and the items that never have interactions with any user in the recommendation. Due to the fact that the majority of user-question interactions in the testing set are users answering new questions, it is questionable to conclude that the pure collaborative filtering model does not work on the MathOverflow data solely based on the observation that it has a quite low AUC score on the testing set. In order to test the effectiveness of pure collaborative filtering, it is a better idea to exclude the new questions from the test set and focus on evaluating collaborative filtering algorithm's performance on merely recommending existing 'old' questions to users. To achieve this purpose, the 6,458 new questions and 21,817 never-answered questions are excluded from both the training and testing set, leaving 49,773 'old' questions in the pool. With the new training and testing set, first a baseline model is established, which is a popularity-based recommender system that ranks the questions based on their popularity (total number of times a question was answered) in the system and recommends the same list of questions to each user. Thus it is a non-personalized recommender and is simply used as a benchmark against which other personalized recommender systems could be compared. The AUC score of the popularity-based recommender system is just 0.4679, which is essentially no better than random. And the AUC score of the pure collaborative filtering model on the new testing set is 0.6770, which is significantly higher than the baseline model as well as the pure collaborative filtering model evaluated on the all-inclusive testing set. This result clearly demonstrates the effectiveness of collaborative filtering in recommending existing questions to appropriate users in spite of the low user activity and participation rate in the system – on average, each user only answered 21 questions, being only 0.04% of total number of questions (total number is 49,773); and each question is only answered by 2 users on average, being only 0.04% of total number of users (total number is 4,513). With such a sparse user-question interactions matrix, collaborative filtering model still managed to learn useful latent user and question features from it which have been proved to improve the recommendation accuracy on the future user-question interactions data. To confirm if the question features provide additional insights to recommendation of existing questions to users, two hybrid recommendation models were built and evaluated on the new testing set. The first one is the collaborative filtering plus the 1,360 tag features of questions, whose AUC on the testing set is 0.8714. The second one is the collaborative filtering plus the 1,360 tag features together with the 50 topical features of the questions, whose AUC on the testing set is 0.8946. The AUC scores of both models are slightly higher than their counterpart models built and evaluated on the entire question pool, providing further evidence for the hypotheses that contents of questions complement the user answering behaviors in terms of enhancing the recommendation accuracy and that collaborative filtering is better at recommending existing active questions (questions answered by users before) to users who have not yet answered them than recommending new or inactive questions to users. Hybrid recommender systems that include user features were not explored for the reduced question pool (with 49,733 existing questions), as such models have already been applied to the entire question pool and the results clearly suggest that user features are not useful for recommending questions on MathOverflow.

Extensions & Applications

In conclusion, in this preliminary case study, hybrid recommender systems were built using the historical question asking/answering data on MathOverflow which allows for personalized recommendation of questions to potentially interested users. Both collaborative filtering and content-based filtering functionalities are enabled in the hybrid model, which are demonstrated to be complementary to each other and simultaneously contribute to improving the accuracy and robustness of personalized question recommendations. Particularly, collaborative filtering is able to identify the latent user & question factors that are closely related to the score of a question in the recommended list to a user but are hidden in the collective user behavior data, whose performance will improve if there are increasingly more user-question interactions within the system; while content-based filtering has the advantage of recommending new questions to existing users and recommending existing questions to new users with high accuracy over collaborative filtering. Also, it was demonstrated that underlying features of the raw texts associated with the items and users could be extracted using text mining techniques, and the extracted features could be potentially useful for boosting the model performance.

The hybrid recommender system discussed in this case study could be further improved in several ways. First, better feature construction strategies and procedures could be designed to clean the unstructured raw text data so as to generate more reliable and meaningful features from topic modeling, as discussed in previous sections. Second, more time could be spent on tuning the hyper-parameters of the model to achieve optimized recommendation accuracy. But this is usually a very lengthy and time-consuming process: there are several important hyper-parameters which could affect the model performance, including the dimensionality of the latent factor vectors of question and user, the regularization strength which is used for alleviating overfitting problems, the maximum number of negative samples used in optimizing the WARP loss function, the number of training epochs which stands for the number of times the entire training set are passed through the model during the training phase, and so on. We might have some prior knowledge about the default values of those hyper-parameters to get a good start, but in order to find out the set of hyper-parameters which result in optimal prediction accuracy on the testing set, all possible combinations of hyper-parameter values need to be explored, and each time when a new set of hyper-parameters is tried, the model has to be re-trained and re-evaluated, which is nontrivial task for the computer. Lastly, if we were to put the MathOverflow recommender system into practice, there will be lots of practical considerations associated with it. One example is the selection of a reasonable value for k, which is the total number of questions the system will recommend to each user. The difficulty of it has already been discussed in previous section. Another example is to define an appropriate metric to evaluate the model performance as the recall at k, precision at k and AUC are a little too simplistic and general and could only provide partial information about the prediction accuracy.

The principle and core methods of the hybrid recommender system for MathOverflow described in this summary can also be applied to build recommender systems for other purposes, such as a job search/application system which leverages user profiles (resume, educational background, job preference, etc.), job posting contents and the online user behavior (click/apply) data to provide personalized recommendation of active job opportunities to individual users based on the hybrid model, or an online learning/sharing system which recommends self-learning courses or study materials to users that best match their interests and capabilities and will benefit their future careers. One nice feature about the recommender systems is that as time goes by, if there are more and more users and items entering the system and their interactions keep increasing, the model could be updated frequently to accommodate new data and continuously improve its recommendation performance, as the more interactions data we get, the more accurate the predictive model is likely to be (especially for the collaborative filtering functionality). As the recommendation model becomes increasingly more reliable, it will in turn provide insights into which user/tem features are useful for improving recommendation accuracy and which are not, and even promoting the design of new features in the user interface of the system so as to facilitate the collection of more informative and useful data from users and items. And because of the dynamic nature of the recommender systems, today’s new data that are being tested will become part of the historical training data tomorrow, and there will always be new data entering the system that could be tested to evaluate the model performance. This opens up plenty of opportunities for experimenting with new system & user interface design and enhancing the recommendation power of the system.

P.S. The link to the GitHub repository where the codes written for this case study are stored:

https://github.com/yifengsong1989uva/Recommender-System-for-MathOverflow-Users