Recommending Tasks in Online Judges using Autoencoder Neural Networks

Paolo FANTOZZI\textsuperscript{1,2}, Luigi LAURA\textsuperscript{1,3}

\textsuperscript{1}Italian Association for Informatics and Automatic Calculus (AICA), Italy
\textsuperscript{2}Sapienza University of Rome, Italy
\textsuperscript{3}International Telematic University Uninettuno, Italy

e-mail: fantozzi@diag.uniroma1.it, luigi.laura@uninettunouniversity.net

Abstract. Programming contests such as International Olympiads in Informatics (IOI) and ACM International Collegiate Programming Contest (ICPC) are becoming increasingly popular in recent years. To train for these contests, there are several Online Judges available, in which users can test their skills against a usually large set of programming tasks.

In the literature, so far few papers have addressed the problem of recommending tasks in online judges. Most notably, as opposed with traditional Recommender Systems, since the learners improve their skills as they solve more problems, there is an intrinsic dynamic dimension that has to be considered: when recommending movies or books, it is likely that the preferences of the users are more or less stable, whilst in recommending tasks this does not hold true.

In order to help the learners, it is crucial to recommend them tasks that are challenging but not unsolvable compared with their current set of skills. In this paper we present a Recommender System (RS) for Online Judges based on an Autoencoder (Artificial) Neural Network (ANN).

We also discuss the results of an experimental evaluation of our approach in both the scenarios in which we consider, or not, the intrinsic dynamic dimension of the problem. The ANNs are trained with the dataset of all the submissions in the Italian National Online Judge, used to train students for the Italian Olympiads in Informatics.

Keywords: autoencoder neural networks, recommender systems, programming contests.

1. Introduction

Programming Contests (PCs) are competitions in which participants are faced a set of tasks that require writing computer programs. Recent literature have emphasized the importance and the effectiveness of PCs in the process of learning computer programming (Audrito \textit{et al.}, 2012; Astrachan, 2004; Blumenstein \textit{et al.}, 2008; Dagienè, 2010; García-Mateos and Fernandez-Aleman, 2009; Wang \textit{et al.}, 2011).

In order to train for PCs, learners use Online Judges (OJs), also known as Programming Online Judges, i.e., web based e-learning tools where a user can submit solutions
to a programming task. The user chooses a task from the many available; after reading its statement, that includes the required formatting of input and output or the use of a programming interface, the user writes a code to solve the task. The code is submitted to the OJ, that verifies both the correctness, usually by testing it against a certain number of test cases, and the efficiency, by checking that the running time and/or the memory usage is under some limit. In Fig. 2 is shown an example of a programming task.

However, choosing the right task is becoming a complex problem, and an example of an information overloading scenario, as observed in Yera Toledo et al. (2018): an unexperienced user has to choose from thousands programming tasks, many of which are probably beyond his current abilities. For example, University of Valladolid Online Judge has more than 200k users and 2k tasks, whilst SPOJ accounts approximately 600k users and 6k (public) tasks; in Fig. 1 we can see the list of available problems in the Peking University Online Judge http://poj.org.

With so many available tasks, it is important to help users selecting their next task by using a Recommender System (RS). Traditionally, RS are broadly divided into two categories: Content Based ones, in which the recommendations derive from features of the items to be suggested, and Collaborative Filtering approaches, in which the suggestion is based on the items chosen by users similar to the current one.

As observed in (Audrito et al., 2019), there are some peculiarities of Online Judges that prevent the use of a general Recommender System:

- Users slowly improve their abilities, one task after the other, so the general concept of user preferences does not apply: recommending a movie or a novel differs significantly from recommending a task; a user will probably still like a novel after one year, whilst he might find a task too easy after the same amount of time.
- Users with similar skills, i.e. users to whom we might want to suggest the same set of tasks, might behave very differently in OJs, thus preventing us from considering them similar. For example, one might solve all the tasks involving a given

![Fig. 1. The list of available problems in the Peking University OJ.](image-url)
Olimpiadi Italiane di Informatica 2014
Fisciano, 18 – 20 settembre 2014
taglialegna • IT

Abbatti S.p.A. (which is the Italian brand of tearDown INC) is a big enterprise that works in the field of tree felling. In particular, it’s been a few years since it started improving in tearing down barky trees, a peculiar kind of trees which is very tall and thick. This particular species grow in a very tidy way: the woods made of these trees are actually a long horizontal line of trunks, placed at one decameter (32, 8 feet) one another. Each one of the trees has a particular height, which is expressed by a positive number (decameters).

Tearing down one of these trees is a very hard thing to do and, although tearDown INC employs the most advanced technologies on the market, it is a very time consuming activity, since barky trees’ bark is incredibly thick. The workers have the opportunity to chose in which direction (left or right) the tree should fall after the cut.

Each time a barky tree falls down it hits the trees that haven’t been torn down which are placed on its falling trajectory; in other words, it tears down each tree which is closer than its height in the direction of the fall. Since the number of barky trees in this woods is huge this dynamic of the fall creates a domino effect.

In order to be the best enterprise in barky tree felling tearDown INC developed a system which is able to scan the whole wood, choosing which trees should be cut by workers and in which directions they should fall with the aim of cutting all the trees. It’s important to recall that it’s in the best interest of the enterprise to minimize the number of trees that need to be torn down by workers directly. Your role in this situation is to implement the system for tearDown INC.

Below we can see an example of a solution of the instance shown in the picture above: it is enough to cut two trees:

Fig. 2. An example of a problem from a programming contest; this task is taken from the final contest of the 2014 edition of the Italian Olympiads in Informatics (OII).
Most notably, as opposed with traditional Recommender Systems, since the learners improve their skills as they solve more problems, there is an intrinsic dynamic dimension that has to be considered: when recommending movies or books, it is likely that the preferences of the users are more or less stable, whilst in recommending tasks this does not hold true. In this paper we propose a task recommender system based on an Autoencoder Neural Network (ANN); in particular, we address both the static case, in which the user is represented by the task he solved, and the dynamic case, where we try to represent the growth of a user by the sequence of the problems he solved. For both cases we present a Recommender System (RS) for Online Judges based on an Autoencoder (Artificial) Neural Network (ANN). We trained and tested the ANN using data from the Online Judge used in the Italian Olympiads in Informatics (Olimpiadi Italiane di Informatica – OII) (Di Luigi et al., 2016), targeted at secondary school students training. We compared our approaches against state of the art more classical recommender systems built using the Simple Python Recommendation System Engine (SurPRISE – http://surpriselib.com). The experimental results confirm the effectiveness of our approach.

Preliminary versions of this paper appeared in the Proceedings of the 17th International Conference on Distributed Computing and Artificial Intelligence (DCAI 2020) (Fantozzi and Laura, 2020a) (the static case), and in the Proceedings 13th International Workshop on Social and Personal Computing for Web-Supported Learning Communities (SPeL 2020) (Fantozzi and Laura, 2020b) (the dynamic case); the comparison against state of the art more classical recommender systems has not appeared before.

This paper is organized as follows: the next section provides the necessary background related to programming contests, online judges, and recommender systems, whilst our approach is detailed in Section 3. In Section 4 we detail our experimental findings and concluding remarks are addressed in Section 5.

2. Related Works and Background

In this section we discuss related work and the necessary background concerning programming contests, online judges, and recommender systems.

2.1. Programming Contests and Online Judges

A programming contest is a competition in which contestants are faced with a set of programming tasks, also called problems, to be solved in a limited amount of time and/or with a limited amount of memory usage.

A single task can be broken into different subtasks of increasing complexity: basic techniques might be enough to solve, within the given time and/or space limits, some of
the subtasks whilst the most difficult ones might require very specific algorithmic techniques and data structures. Popular programming contests are:

- The International Olympiads in Informatics (IOI), that are an annual programming competition for secondary school students patronized by UNESCO. http://www.ioinformatics.org/
- The ACM International Collegiate Programming Contest (ICPC) is a multitier, team-based, programming competition operating under the auspices of ACM. https://icpc.baylor.edu/
- The very recent International Olympiads in Informatics in Team (IOIT), that started in 2017, that are a team competition, like ACM ICPC, differently from IOI (individual competition). Currently there are only four nations involved: Italy, Romania, Russia, and Sweden. https://ioi.team/
- Google Code Jam, that is based on multiple online rounds that concludes in the World Finals. https://code.google.com/codejam/
- Facebook Hacker Cup, that is (citing from their site) “an annual worldwide programming competition where hackers compete against each other for fame, fortune, glory and a shot at the coveted Hacker Cup”. https://www.facebook.com/hackercup/

The Online Judges are, usually, web based platforms that provide a large number of programming tasks to be solved. There are several popular OJ platform, we cite the already mentioned University of Valladolid Online Judge https://uva.onlinejudge.org, Sphere Online Judge (SPOJ) https://www.spoj.com/, CodeChef https://www.codechef.com/, and Peking University Online Judge http://poj.org.

Yera and Toledo (Yera Toledo et al., 2018) present a brief survey on OJs, whilst more information on tools and techniques for automatic evaluation of solutions submitted to OJs can be found in (Ala-Mutka, 2005; Caiza and Del Alamo, 2013).

2.2. Recommender Systems in OJs

As already observed in the introduction, despite the large amount of literature devoted to RS, the peculiarities of recommendation in OJs, where the relation user-item is way more complex than the typical RS cases, prevent from using standard techniques and forces the development of ad-hoc methods. This aspect is detailed in the paper of Audrito et al. (2012), where the authors propose a first approach on building a RS by tackling the problem of ranking tasks in Online Judges.

Indeed, so far few research focused in the recommendation of tasks in OJs: we mention the traditional collaborative filtering method with a new similarity measure adapted to the case (Toledo and Mota, 2014), and an approach based on fuzzy logic (Yera Toledo et al., 2018). Caro and Jimenez considered user-based and similarity-based approaches In (Caro-Martinez and Jimenez-Diaz, 2017). Di Mascio et al. proposed a framework that can allow recommendations and that can foster motivation in students by means of a lightweight, badge-based, gamified approach (Di Mascio et al., 2018).
There is an online tool, developed by Stephen and Felix Halim, authors of the book Competitive Programming (Halim and Halim, 2013), called uHunt, that helps its users to choose the next problem to be solved, as shown in Fig. 3: their very practical (and effective) approach is to rank the problems according to their dacu, i.e. the distinct accepted users. Indeed, as they state, “The bigger the dacu the easier the problem should be and the more probable it will appear in the UVa discussion board”.

The “classical approaches” that use counting to estimate the grade of difficulty have a well known drawback: the items suggested to the users will be always the popular ones that will become even more popular, and so even more recommended. This means that a new item will never be suggested.

2.3. Recommender Systems and Artificial Neural Networks

The use of deep learning techniques for recommender systems is divided in two categories that we call classical and hybrid.

The classical approach uses the standard architectures of neural networks, applying them to this task. So, in this case, the most important part consists in the formulation of the problem, since that, if the input data are not suitable to be the input of that specific deep learning technique, then the result will be totally inaccurate.

The other approach is the hybrid one, that consists in using more than one type of architecture at the same time. This kind of approach is useful when the input data are
not easily representable as a standard structure, like a user-item matrix. In (Zhang et al., 2016) the authors use a Convolutional Neural Network to extract features from images and then an Autoencoder to build the recommender system on the features.

In (Zhang et al., 2017a), in order to build a recommender system for hashtag in tweets, the authors use at the same time some CNNs and some Recurrent Neural Networks (RNNs). In this work they use the CNNs to extract features from the image and then the RNNs to extract information from the text, combining them using different weights based on co-attention.

Since that a recommendation task is similar to a dimensionality reduction task, many of the state-of-the-art techniques use some kind of Autoencoder to map the input in a smaller space, that will be the representation of the correlations in the recommender system. In particular, Sedhain et al. (2015) introduce the using of a vanilla Autoencoder to build a recommender system. They use a partial masked input (the same techniques we use in this work) and try to reconstruct it in output, the elements added in the output will be the recommended elements. Strub and Mary (2015) extend the work of Sedain et al. (2015): they use a denoising Autoencoder instead of the vanilla Autoencoder to build a more robust system.

Chen and de Rijke (Chen and de Rijke, 2018) follow a similar approach, but they use a Variational Autoencoder to perform top-N recommendation. They encode both the user ratings and some side information in the compact space in the Autoencoder. Zhang et al. (Zhang et al., 2017b) generalize the Contractive Autoencoder paradigm into matrix factorization framework. Li et al. (Li et al., 2015) combine a probabilistic matrix factorization with Marginalized Denoising Stacked Autoencoders to perform collaborative filtering. This work can be considered as a general framework to use these kinds of techniques; in this context, several works, including (Van den Oord et al., 2013; Wang et al., 2015; Wang and Wang, 2014), can be viewed as special cases of this framework.

3. Recommending Tasks Using Autoencoder Neural Network

Our goal is to provide recommendations to the users regarding the next task to deal with among all the tasks in a system. To build this kind of system we assume that:

- If a user obtains a score for a task it means that it is the max score possible for that user in that task.
- A user should solve the problems sorted by their grade of difficulty for the user; thus, a user should never try to solve a problem that is much harder than the last one he solved.
- It is possible to deduce the score for a new problem based on the scores the user obtained in other problems.

Note that there is one more assumption that is valid for the static case but not for the dynamic: given a snapshot of the scores for the users it is not important the order followed by the user to solve the tasks to foresee the score for another task.
Based on the above assumptions, we decided to build a model that takes as input the current scores of a user and provides probable scores for the same user for other tasks. Then we can choose between the forecasted scores and suggest the task with the highest score between them.

If we consider the score as a judgment of the user for the item (i.e., the programming task), then we can just exploit the already known techniques for recommending items. We chose to use an Autoencoder to build the model. Since that we want to use just the scores of the users, without any information from other sources, we take as input the scores for all the tasks and we differentiate between static and dynamic:

- Static: we mask a fraction of the scores in input as a non-solved task and we perform back-propagation from the complete scores.
- Dynamic: we use the scores of a fixed moment in time as input and we perform backpropagation from the next moment.

In this way, in the bottleneck layer, there should be a compact representation of the similarities of the tasks.

4. Experimental Evaluation

In this section we describe the results of our experimental evaluation. We distinguish the two approaches, i.e. static and dynamic, in the next sections, and then compare the results against the ones obtained using a state of the art more classical recommender system built using the Simple Python Recommendation System Engine (SurPRISE – http://surpriselib.com).

4.1. Autoencoder Neural Networks: Static Case

To test the method we have designed, we have taken the submission to the OII Training platform (Di Luigi et al., 2016) in a defined time range. The submissions were in the form:

< user_id, task_id, datetime, score >

where each submission corresponds to a possible solution to a task from a user that performs a certain score, based on many test cases. We filtered out all the scores equal to zero because we can't know if they were just users testing the behaviour of the platform. Then we considered only the best score for each task, for each user, to ignore all the attempts to solve the problem before the user found the solution.

We performed a preliminary set of experiments with the original data: we built a user x task matrix where each cell contains the best score of the user for the task. The result is a 3148 x 409 matrix with 43051 non-empty cells. The matrix has an average of 105 submissions for each task and 13 for each user. The max number of users which have
submitted to the same task is 1070 and the max number of tasks with submissions from the same user is 336. We consider the zero valued cells as a problem with no submission from the user.

To use this matrix as a training set for this model, we duplicated the matrix and then we have randomly masked some positive scores with zero. The masked matrix will be the input to the model and the original matrix will be the output to reconstruct. The number of tasks masked for each user is a random number between 3 and 7, with the constraint that it should be anyway equal at most to the half of the submission for the user.

After the preliminary experiments, it was clear that data was too small, thus we performed an operation of data augmentation. In particular, we have repeated many times the same rows of the matrix with the result of a matrix with 8 times the rows of the original. Then each row has been randomly masked independently, so we unlikely had duplicated rows in the matrix. We have loaded all the data on a Google Colab instance with an available GPU. Then we have split the data on train and test set with a ratio of 0.8/0.2. We used Tensorow to build several models; the smallest was a Sequential model with 5 layers: the input layer, a dense 64 neurons layer, a dense 16 neurons layer, a dense 64 neurons layer, and an output layer with dimension equal to the input layer. All the activations for the layers are ReLU with a constraint of a max value of 1.0 (the max value of the score). We used an Adam optimizer with a learning rate of 0.001 and a mean squared error loss function.

We trained the model for 500 epochs with a batch size of 128, and we have imposed a validation split of 0.2. The resulting learning curve is shown in Fig. 4, whilst the accuracy curve measured is depicted in Fig. 5.

The standard accuracy might be not fully representative of the error of the model (since that we have a sparse matrix), thus we computed a sum of the squared errors on each samples in the test set. The resulting values follow the distribution shown in Fig. 6; in Table 1 we report some stats of the SSE distribution.

![Model loss](image)

*Fig. 4. Model loss – static case.*
Fig. 5. Model accuracy – static case.

Fig. 6. Distribution of sum of squared errors (SSE) – static case.

Table 1
Statistics of the distribution of SSE (Fig. 9) – static case

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>2.27</td>
</tr>
<tr>
<td>std deviation</td>
<td>6.07</td>
</tr>
<tr>
<td>min</td>
<td>0.00</td>
</tr>
<tr>
<td>25% (1st quartile)</td>
<td>0.00</td>
</tr>
<tr>
<td>50% (2nd quartile)</td>
<td>0.05</td>
</tr>
<tr>
<td>75% (3rd quartile)</td>
<td>1.38</td>
</tr>
<tr>
<td>max</td>
<td>66.19</td>
</tr>
</tbody>
</table>
4.2. Autoencoder Neural Networks: Dynamic Case

In the dynamic case, our interest was in the evolution of the users, so the dataset that we used was derived from the one, described in the previous section, for the static case. Indeed, from the baseline dataset, we built a user x task matrix where each cell contains the best score of the user for the task. The result is a $42155 \times 409$ matrix with 2035447 non-empty cells (we had to drop few submissions from the baseline dataset). As before, the dataset has an average of 105 submissions for each task and 13 for each user. The max number of users which have submitted to the same task is 1070 and the max number of tasks with submissions from the same user is 336. As before, we consider the zero cells as a problem with no submission from the user.

Since that the resulting matrix has many rows for each user (one row for each problem solved) that contains all the scores of the users until that moment, we use a row as input and we impose the next row for the same user as output. This means that we consider $n_i - 1$ samples for each user $i$, where $n_i$ is the number of problems solved by the user $i$.

As for the static case, we load all the data on a Google Colab instance with an available GPU and the data was split into train and test set with a ratio of 0.8/0.2. We used Tensorflow to build several models; the smallest was a Sequential model with 11 layers: the input layer, two dense 128 neurons layer, two dense 64 neurons layers, a dense 32 neurons layer, two dense 64 neurons layer, two dense 128 neurons layers, and an output layer with dimension equal to the input layer. Also in this dynamic case, all the activations for the layers are ReLU with a constraint of a max value of 1.0, and we used an Adam optimizer with a learning rate of 0.001 and a mean squared error loss function. We trained this model for 100 epochs with a batch size of 128, and we have imposed a validation split of 0.2.

The resulting learning curve is the shown in Fig. 7, whilst the accuracy curve measured is depicted in Fig. 8.

![Model loss](image)

**Fig. 7.** Model loss – dynamic case.
Since that the standard accuracy doesn't represent well the error of the model (i.e., we have a sparse matrix) we computed a sum of the squared errors on each samples in the test set.

The resulting values follow the distribution shown in Fig. 9; in Table 2 we report some stats of the SSE distribution.

Overall, from the results shown above, it seems that the static approach, described in the previous section, seems to perform better than the dynamic one; this might be due to the way the dataset has been built, and we plan to compare the two approaches against other different datasets.
4.3. **Comparison Against Classical Recommender Systems**

In this section we compare our approaches against state of the art Recommender Systems built using the python SurPRISE library: we used 11 models from this library. We trained these models using the static dataset, i.e. the original dataset; the dynamic dataset derives from this one and the dynamic ANNs, as seen in the previous section, performed not as well as the ones trained on the static dataset.

Indeed, we experimented with 32 different ANNs for the static case and, after evaluating the best performers, we experimented with 8 different ANNs for the dynamic case. In Table 3 we can see the mean square error (MSE) for all 11 SurPRISE models,

<table>
<thead>
<tr>
<th>Model</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NormalPredictor</td>
<td>0.1004863793</td>
</tr>
<tr>
<td>CoClustering</td>
<td>0.0903564508</td>
</tr>
<tr>
<td>SlopeOne</td>
<td>0.0599415086</td>
</tr>
<tr>
<td>NMF</td>
<td>0.0545891728</td>
</tr>
<tr>
<td>KNNBasic</td>
<td>0.0544956720</td>
</tr>
<tr>
<td>KNNWithZScore</td>
<td>0.0528847039</td>
</tr>
<tr>
<td>KNNWithMeans</td>
<td>0.0526918100</td>
</tr>
<tr>
<td>SVD</td>
<td>0.0512103173</td>
</tr>
<tr>
<td>BaselineOnly</td>
<td>0.0484940726</td>
</tr>
<tr>
<td>SVDpp</td>
<td>0.0479013953</td>
</tr>
<tr>
<td>KNNBaseline</td>
<td>0.0478215959</td>
</tr>
<tr>
<td>(dynamic) autoencoder-plus-time-512-128-Dropout(0.1)-512-lr0.001</td>
<td>0.0031555248</td>
</tr>
<tr>
<td>(dynamic) autoencoder-plus-time-2048-512-Dropout(0.1)-2048-lr0.001</td>
<td>0.0025685430</td>
</tr>
<tr>
<td>(dynamic) autoencoder-plus-time-1024-256-Dropout(0.1)-1024-lr0.001</td>
<td>0.0025619145</td>
</tr>
<tr>
<td>(static) autoencoder-2048-512-Dropout(0.1)-2048-lr0.0005</td>
<td>0.0000349358</td>
</tr>
<tr>
<td>(static) autoencoder-1024-256-Dropout(0.1)-1024-lr0.0001</td>
<td>0.0000281454</td>
</tr>
<tr>
<td>(static) autoencoder-2048-512-Dropout(0.1)-2048-lr0.0001</td>
<td>0.0000191324</td>
</tr>
</tbody>
</table>
and for the three best ANNs for both the static and the dynamic case. The results shown in the table are sorted from the biggest (worst) to the smallest (best). It seems that, at least for this dataset, the ANNs outperform each model from the SurPRISE library. In the table, the three best results belong to ANNs trained for the static case, but in all our experiments, i.e. 40 different ANNs (32 for the static case and 8 for the dynamic case), we did not observe such a clear separation between the ANNs trained with the two different datasets.

5. Conclusions

In this paper we proposed the design of a recommender system for tasks suggestions in Online Judges, based on a Autoencoder Neural Network. We trained the ANN with the data from the OJ used by the secondary school students training for the Italian Olympiads in Informatics (Olimpiadi Italiane di Informatica – OII) (Di Luigi et al., 2016; Di Luigi et al., 2018).

We tested two different approaches: a static one, that is more typical of a recommender system, and a dynamic one, in which the dataset has been modified in order to explicitly represent the evolution of a user. We also compared our approaches against more traditional Recommender Systems model built using the python SurPRISE library.

We definitely think that Online Judges deserve their specific recommender systems, and we hope that our one is a first step to the development of such systems. We plan to implement our approach inside the italian OJ, and we are available to collaborate with the developers of other Online Judge systems, by either implementing RS into those systems or by testing our models against other datasets.

References


**P. Fantozzi** is involved in the training of the Italian team for the IOI since 2018. He is a Ph.D. student in Engineering in Computer Science at “Sapienza” University of Rome. He is lecturer at LUISS University for the courses: Lab of computer skills, Customer intelligence and big data analysis logics, Introduction to network science.

**L. Laura** is Associate Professor at Uninettuno university; he is involved in the training of the Italian team for the IOI since 2007, and since 2012 is in the organizing committee of the Italian Olympiads in Informatics. He got a Ph.D. in Computer Science in the “Sapienza” University of Rome.