# Two interaction modules and a hybrid model

Given the information about a mashup to be built m, all selected services SS and a service to be tested s, our framework in section 3 learns their complex interaction and predicts the probability of m selecting m as next service. In this section, we apply the framework to two kinds of information (that is, content information and invocation history between mashups and services) and propose two modules that learn the interaction from different perspectives. We will mainly detail the difference between the two modules, that is, their input information and feature representation. Finally, we propose a hybrid model that integrates these two modules to make a more accurate prediction.

## Functional interaction module

When assessing a service s, a mashup developer will consider whether the functionality of s can satisfy his requirement, Mreq, along with the selected services SS. Therefore, we design a functional interaction module that exploits their content information to model their interaction from a functional perspective.

The functional description of a service, that is, their content information, generally fall into two forms: word sequence (such as description) and separate word set (such as tags). The same is true of Mreq. To get the representation of functionality of a mashup or a service, we first adopt two deep-learning based techniques to process these two forms of information respectively, and then concatenate their extracted features.

To apply deep-learning based feature extraction, we need to preprocess content information with word embedding. Specifically, we convert all terms into sparse binary vectors with one-hot encoding, input these vectors into an embedding layer and map each term into a dense vector or an embedding. Finally, we truncate or pad a piece of content information if necessary and stack the embeddings of the terms in it, transforming the information into a matrix with a fixed size. The process can be described as

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

where denotes the word-embedding representation of the content information, is the length of the processed information, is the -th term in the information, and is the word embedding for .

For word sequence information, we apply *text\_inception* we proposed in [] to its feature extraction. The method first parallels stacked convolution layers to extract local patterns in the word sequence, then employs a global average pooling (GAP) layer to emphasize important features. Finally, an MLP is used to perform non-linear transformation on the features. This process can be simply written as

where denotes the representation extracted from word sequence information.

For content information in the form of a separate word set, there is no order among multiple terms in it, so *text\_inception* is not applicable. Instead, we retrieve and average the embedding of each term to get a fixed-length representation of the information:

where is the size of the set and denotes the representation extracted from word set information.

For a mashup or service, we concatenate the two features extracted from its content information,and, and obtain a representation depicting its functionality:

At this point, we can transform the content information of MReq, s and each selected service into real-valued feature vectors. Then we feed them into the interaction layer and finally obtain an vector, , to model their functional interaction.

## Neighbor interaction module

Besides content information, the historical invocations between existing mashups and services is beneficial to recommendation result. Inspired by user-based CF methods, we design a module named as neighbor interaction module, to learn the interaction among m, SS and s based on the historical information of the mashups similar to m (i.e., neighbor mashups, denoted as below). The hinge of applying our framework to historical information is how to get the representations of m, s and selected services. We will detail it in this subsection.

In order to get the representations of existing mashups and services, we use node2vec,【】 a typical graph embedding method, to process the graph transformed from the historical invocation matrix *MS*. Compared with traditional matrix factorization (MF) based methods, such as PMF and SVD, node2vec is capable of capturing more non-linear relationship between a mashup and a service.

Note that m is built and completed online, so the above model-based methods cannot update their models with the newly selected services, SS, nor get an effective representation of m. Instead, we first look for some neighbor mashups for m, and then compute the representation of m using that of and the similarities between them. Therefore, the key lies in how to calculate the similarity between m and an existing mashup . To this end, we adopt the method proposed by【】. Specifically, we first extract topics from content information by Latent Dirichlet Allocation (LDA) and use top3 topics to represent the functionality of a mashup or service. Then a heterogeneous information network (HIN) containing two kinds of objects (mashup and service) is constructed. We use a meta-path-based method to calculate six types of associations or similarities between two mashups: having the same topics, labelled by the same tags, co-invoke one service, co-invoke similar services that have some same topics, same tags or the same provider. Distinct weights are assigned to each similarity, and the weighted sum of these six similarities are calculated as an overall similarity between two mashups. We set the weights to the values pre-trained by【】. The whole process to calculated an overall similarity between and **,** , can be expressed as：

|  |  |
| --- | --- |
|  | (11) |

where is the p-th meta-path-based similarity**, and** is its weight.

According to the similarity between m and each existing mashup, we can select most similar mashups as NM, and then get a weighted representation of ,:

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

where is the representation of obtained by node2vec.

In addition to the content information between two mashups, this similarity calculation method also takes into account the information of their component services. Each time when a developer selects some new services, the method improves its similarity measurement and gets a better picture or representation of the new mashup. The meta-path-based method to calculate mashup similarity is also efficient. If necessary, we can adopt some pruning strategies, that is, set some rules to reduce the size of candidate mashups, to further improve the efficiency of looking for neighbor mashups.

Up to now, we have obtained the representations of m, each service in SS, and s in the same feature space based on the historical information. Then we can input them into the interaction layer and compress their interaction in this space into a dense vector,.

## Hybrid model

The above two modules learn different forms of interaction among m, SS and s based on their content information and historical information respectively. Below, we propose a hybrid model that integrates these two modules to exploit multiple interactions. Its architecture is shown as Figure 2.

Our hybrid model consists of two underlying modules, an integration layer and a predict layer. Each underlying module is constructed with the framework we proposed in the section 3. It receives a kind of information and learns a unique kind of interaction among m, SS and s. Then, the integration layer [incorporate](https://www.dictionary.com/browse/incorporate)s all interactions learned from different underlying components with an MLP. Finally, the output layer makes a more accurate prediction.

Our hybrid model is extensible. If a new form of information is available, we can first add another underlying module to learn a new kind of interaction, and then easily integrate it into the original model to improve model performance.

## Offline Model Learning

Our model predicts the probability of a mashup m choosing a service s as next service when it has selected some services SS, so a sample in our study is denoted as . The predicted value should approximate 1 for a positive sample and 0 for a negative sample, so the likelihood function of our model is:

(19)

where θ denotes all model parameters, is model’s predicted probability for a sample , and denotes positive sample set and negative sample set respectively.

To maximize the above likelihood is equivalent to minimize the following loss function:

(20)

where is the actual label of , and it equals 1 for positive sample and 0 for negative sample.

Based on the above loss function, we employ Adam algorithm [] to update all model parameters. Since our hybrid model is a hierarchical model, directly updating all parameters may result in slow convergence. Therefore, we first train each underlying modules and then train the integration layer and the output layer. On this basis, we fine-tune the whole model finally.

Every time when SRS has served developers to build sufficient mashups or periodically (e.g. every other week), we can update the model based on the training samples generated from these new mashups and their component services.