# Two interaction modules and a hybrid model

Given a mashup *m* to be built, 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 s as the 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 models that learn the interaction from different perspectives. We will discuss the differences between the two models including their input information and feature representation. Finally, a hybrid model that integrates these two models is introduced.

## Functional-interaction-based model

When assessing the selection probability of a service *s*, a mashup developer will firstly consider whether the functionality of *s*, along with those of selected services *SS*, can satisfy his requirement *Mreq*. Therefore, we design a functional interaction model that captures their interaction from the perspective of functionality.

The functionality of a service, that is, the content information, generally fall into two forms: word sequence (such as service descriptions) and separate words (such as tags). The same is true of Mreq. To get the representation of the 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 the 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 its terms, which transform the content information into a matrix with a fixed size,. The process can be described as

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

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

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

For the content information in the form of a separate word set, there is no order among its multiple terms, which makes *text\_inception* inapplicable. 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.

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

In this way, 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 obtain an vector, , to model their functional interaction. Finally, we directly input this vector into a softmax layer to predict a score.

## Neighbor-interaction-based model

Besides the content information, historical invocations between existing mashups and services is beneficial to the recommendation result. Inspired by user-based CF methods, we design a 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 ). The hinge of applying our framework to historical information is how to get the representations of m, s and selected services.

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 can capture 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 the content information by Latent Dirichlet Allocation (LDA) and use top three topics to represent the functionality of a mashup or service. Then a heterogeneous information network (HIN) containing mashups and services is constructed. We use a meta-path-based method to calculate six types of associations or similarities between two mashups: sharing the same topics, labelled by the same tags, invoking the same service, and invoking similar services that have the same topics, tags or providers. Different 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 overall similarity between and **,** can be expressed as：

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

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

According to the similarity between m and each existing mashup, we select most similar mashups as , 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,. The vector is fedinto a softmax layer to generate a prediction.

## A Hybrid model to incorporate different interactions

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

Our hybrid model consists of two underlying interaction modules, an integration layer and an output layer. Composed of a feature extraction layer and an interaction layer, each underlying interaction module is indeed the model we proposed in the previous two subsections without output layer. 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. The process can be expressed as

Finally, the output layer makes a prediction based on the learned comprehensive interaction.

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 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 samples and negative samples, respectively.

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

(20)

where is the actual label of .

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