Mmashup API背景，服务推荐的大致介绍。

现有研究的场景不太对，提出值得研究的新场景，场景分析和需要注意的关键点，现有的CF和content的方法都有缺陷。

提出一种框架：三者交互。Attention。

内容信息和历史信息，都可以使用这种框架。方法实例。实例整合。

1. 提出新场景，场景分析；
2. 提出新框架；
3. 各种信息使用本框架的实例化，各种实例的整合
4. 实验效果

开发者组合多个服务，搭建一个mashup。

现有的服务推荐技术大都直接借鉴传统的电商推荐算法。传统电商的推荐，**离线**分析用户对项目的反馈信息（隐式或显式），挖掘用户隐含的兴趣，然后再计算用户和item的匹配程度。它们首先把mashup开发者已经调用的服务作为其历史记录，然后离线对其兴趣进行建模，最后在线为其推荐可能感兴趣的服务。

但是在服务推荐领域，实际的应用需求，以及理想的应用场景，和传统电商推荐的场景有很大的不同。 ~~直接这么僵硬的移植，存在一些严重的问题~~。

我们认为，现有大部分的服务推荐算法针对的是静态的场景，即，它离线对一个有一些组件服务的mashup建模，然后一次性地为这些mashup推荐服务。而不能在真实的在线交互式的场景中使用，为在线声明需求和动态选择组件的mashup提供服务。

Most of the existing service recommendation algorithms are aimed at static scenarios, that is, they model the mashups with some component services offline, and then recommend some other services for these mashups at once. However, they can hardly work in real online interactive scenarios where mashups provide their requirements online and dynamically select component services.

我们根据开发者真实的体验，归纳出开发者开发mashup的搭建流程：当开发者试图搭建一个新的mashup时，首先向服务推荐系统中输入其功能需求。然后，推荐系统分析其需求并为其生成一个服务列表，接着开发者便会根据需要，从中选择若干个组件服务。如果推荐结果不幸没有完全满足开发者需要，那么系统需要根据用户需求和已选择的services继续生成新的服务列表供开发者选择。这个过程持续进行，直至开发者需求被完全满足，mashup搭建完成。

According to the real experience of mashup developers, we summarize an ideal process of developing a mashup. Suppose a developer wants to build a new mashup and enters his functional requirements () into the service recommendation system (SRS). SRS then analyzes the requirements and generates a list of candidate services, from which the developer selects several component services as needed. If the recommendation result does not fully meet the requirements of the developer, SRS needs to continue to generate a new service list for the developer based on the requirements and the selected services. This step loops until the mashup is set up ~~requirements of the developer are fully met.~~

The mashup developers complete the mashup construction online with the interaction with SRS.

This scene of mashup development is characteristic and has some special needs for SRS.

A good recommendation system should be clear about user needs. For the above development needs, SRS system needs to pay attention to some issues.

用户在与推荐系统交互的过程中，**动态**地完成mashup搭建。这个开发场景有以下特点：

1. 整个搭建过程是一个session or transaction。开发者搭建一个mashup的目的是实现一些特定的功能，这个过程中它关注的是如何实现当前的短期目标，即如何调用多个服务以利用他们的功能，同时开发者也会注意到多个服务间的关系。它之前开发过的mashup对它的影响不大。

The entire process to build a mashup is a session or transaction, whose purpose is to achieve some specific functions by reusing several services. In this process, the developer focuses on how to realize the current short-term goal, that is, how to select the component services in terms of their functions and their relationship. ~~The mashups the developers set up before have little impact on the session.~~

Therefore, next service to be selected is supposed to coordinate with selected services to meet developer requirements. This requires SRS to consider the complex interaction among the mashup to be built, the selected service and next service when making recommendations.

~~换句话说，一个开发者可能之前开发了多个mashup，但是就当前开发的mashup而言，它在开发时主要考虑当前的目标，而之前开发过的mashup对它的影响不大。~~

~~当然，开发过程可能被一些其他因素影响，比如，开发者长期以来对特定提供商提供的服务的偏爱，但是我们暂时不考虑哪些次要因素。~~

1. 用户搭建一个新的Mashup的session，完全是在线的，交互式的。整个开发过程中，用户进入系统输入它的需求，系统在线分析给出结果，用户从结果中选出满意组件，然后系统即时基于这个反馈，在线更新推荐结果。

The session to build a new mashup is completely online and interactive. ~~The user enters the system and input its requirements, the system analyzes the requirements and makes recommendation online, the user selects some components from the results as needed, and then the system updates the recommended results based on the feedback and the requirements of the user in real time.~~

The developer input its requirements and select some component services online.

This requires SRS to interact with developers online. For newly-established online requirements, SRS should make personalized recommendations for it online. After the developer selects some component services, SRS ought to generate new recommendations by integrating user requirements and these real-time feedbacks.

~~最开始待搭建mashup时没有选择服务，推荐系统便无法利用其历史记录。而在用户在线选择一些服务之后，基于模型的推荐技术，也很难利用这些新选择的服务，即时更新其模型。~~

在这个过程中，服务推荐系统需要在下面两种情况下工作：(1)当开发者未选择服务时,(2)在用户已经选择一部分服务时。第一种情况下，推荐系统面临着典型的用户冷启动问题，已有很多相关研究。本文将研究第二种情况，即，在开发者**在线**构建mashup的过程中，当他**已经选择了几个service**s（referred as to context services）时，服务推荐系统如何继续为用户推荐services，帮助他完成mashup的搭建？简而言之，已知开发者当前session的需求，以及已经选中的一部分items， 如何推荐next item？

In this process, SRS needs to work in the following two situations: (1) when the developer does not select any component services, and (2) when the developer has selected some components. In the first case, SRS faces the typical user cold start problem, and there have been many related researches【】. We will study the service recommendation in the second case, that is, given the developer's requirements and some component services that have been selected (referred as to context services) in the current session, how SRS recommend next service for the developer to complete the mashup?

一个好的推荐系统应该分析清楚用户的需求。对于上述mashup开发者的开发过程，服务推荐系统需要注意以下几点：

1. 推荐next item时，既要关注开发者当前session的需求，也要考虑已选择的服务。Next item应该同已选择的服务一起，共同满足开发者需求。要考虑待搭建mashup，已选服务和待测服务之间的复杂交互。
2. Therefore, when recommending next service, SRS should pay attention to both the developer's current session requirements and the selected service. Next item should work with selected services to meet developer needs. To consider the mashup to be built, the complex interaction between the selected service and the service under test.
3. 推荐系统可以在线与开发者交互。对于开发者在线上即时建立的开发需求，推荐系统应当在线对该需求建模，进而对它进行个性化推荐。当开发者从推荐结果中选择一些组件服务后，推荐系统也应当即时利用这些新的交互或反馈，生成新的推荐结果。

现在已有的大部分服务推荐技术大都借鉴传统的电商推荐算法，针对传统的服务推荐场景，即离线对开发者建模，在线推荐服务列表。

在这种在线session型的mashup搭建场景中，现有的大部分服务推荐技术都不能满足上述要求，不能很好的工作。

In this scenario of building a mashup in an online session, most of the existing service recommendation technologies cannot meet the above requirements or work well.

基于CF的算法：基于模型的，user-based，item-based

基于内容的

基于模型的协同过滤算法很难在线为新mashup推荐。它们首先需要使用mashup已经选择的组件服务，离线训练一个模型，然后才能在线为这些mashup推荐新的服务。一种流行的框架是，首先训练模型，获取训练集中的mashup和服务在同一个空间的特征表示，然后在serving阶段预测这些mashup对其他服务的评分，并据此生成得分最高的topK个候选作为推荐结果，例如基于矩阵分解算法的模型，神经协同过滤模型等【MF,NCF】。但是在本文的**在线会话式**服务推荐场景中，这种方法遇到了极大的困难。开发者在线即时建立的全新的开发需求，在模型训练阶段尚不存在，无法获取它们的表示。即便理论上可以基于该mashup在线即时选择的组件services获得该mashup的表示，但这种操作需要更新整个模型，十分消耗资源和时间。再考虑到众多的搭建session，以及每个session内开发者增量地选择组件服务，所以这种设想对于在线会话式推荐系统而言是不可忍受的。因此，由于无法获得每个session的待搭建的mashup的表示，这类算法不能应用在在线会话场景中。

It’s difficult for model-based collaborative filtering (CF) algorithms to make recommendations for the new mashups built online.

They need to train their models using the invocation history between some mashups and services offline before they recommend new services for these existing mashups online.

A popular framework is to first train the model and obtain the representations of the mashups and services in the training set offline, and then predict the ratings of these existing mashups on other services during the online serving phase, eg, MF, NMF【】.

But these methods encountered great difficulties in the online conversational service recommendation scenario. The new mashups are built online and we cannot get their representations form the trained model. Even though we can theoretically get their representations by updating the model based on the component services that the mashups select online, it is too expensive and intolerable for online conversational recommendation systems.

user-based的CF方法和基于内容的方法，在这种场景下都可以工作,但是它们都没有正确地对三者之间的交互关系建模。

user-based的方法，利用该session内已经选择的服务和该session的搭建需求，在线为待搭建mashup寻找相似的历史mashup，然后借助近邻mashup与待测服务间的交互关系，推测待搭建mashup与待测服务的交互。基于内容的方法可以选择那些提供的功能与需求的相似度比较高的services，也就是更可能满足需求的services。

总而言之，它们都只考虑了待搭建mashup和待测服务的关系，而没有考虑已选择的服务与待测服务之间的关系，而实际上，已选择的服务对next item有很大影响。

~~而item-based的方法，则是直接为待搭建mashup推荐与已选择的服务相似的服务。~~

混合算法一般是基于内容和基于CF算法的结合，所以也面临着跟上述方法相同的问题。 Both user-based CF and the content-based methods can work in this scenario, but neither of them correctly models the interaction among the three. The user-based CF methods first seek for some existing mashups similar with the mashup to be built online (referred as to neighbor mashups), and then use the interaction between neighbor mashups and services to infer that between the new mashup and services. The content-based methods tend to recommend the services whose functionality is similar with developing requirements. They only consider the relationship between the new mashup and candidate services, but ignore the relationship between context services and next service. In fact, context services has a great impact on the next service.

As a combination of content-based and CF-based algorithms, hybrid algorithms face the same problems as the two algorithms.

**一个总结：**

1. **基于开发者的实际体验，我们提出了一种在线session式的服务推荐的新场景。**这种场景能够改善开发者实际应用体验，加快服务推荐算法的落地。**我们分析了这种场景的特点，总结出它对服务推荐系统的要求，**即，在搭建mashup的session中，服务推荐系统要考虑开发者需求，已选择的服务和next item的共同交互。对开发者在线选择的服务，系统应当实时利用，更新推荐结果。

We propose a scenario of building a mashup in an online session and summarize its requirements for the SRS that work in it. In the session, SRS must consider the complex interaction among the new mashup, the selected services and next service. SRS should react to the services selected online and update the recommendation results.

（2）我们提出了为在线session式的服务推荐设计的一种新模型。它可以对待搭建mashup，已选择的服务，和待测服务间的复杂交互关系建模，并基于此选出能够与已选择的服务兼容，共同满足搭建需求的next item。具体的，这种模型使用了attention机制，能够区分不同的已选择服务对next item的影响力大小。同时这种模型也能够在线上serving。模型中有个part用于整合用户已选择的服务，当用户选择了新的组件服务时，这部分可以对新选择的服务做出反应，改变三者之间的交互关系的刻画，做出新的推荐。

We design a new deep learning based model for online session-based service recommendation. It models the complex interaction using MLP and recommends the services that are compatible with the selected services and satisfy the developing requirements. Specifically, this model uses the attention mechanism to distinguish the influence of different selected services on the next service. The model can serve online: it integrates all selected services and make new recommendation in real time when some services are selected online.

（3）这种模型具有适用性，我们提出了两个使用这个框架的模型实例。具体的，我们分别利用内容信息和mashup与服务之间的调用关系信息，得到了待搭建mashup，已选择的服务，和待测服务在两个不同的向量空间的特征表示。在每个空间上，我们都可以使用这种模型，对他们的交互关系建模并作出预测。这种模型也具有很好的组合性。我们将上述两个实例进行结合，得到了一个混合模型，它可以结合不同信息空间的交互，对三者之间的交互关系更准确的建模，生成更精准的推荐结果。

We propose two model examples using this framework that is applicable to various domains. Specifically, we obtain the feature representations of new mashup, the selected services and next service in two spaces using the content information and the invocation between mashups and services respectively. In each space, we capture their complex interaction and make a prediction by this model.

This model is also very combinable. We combine the above two models and obtain a hybrid model. We combine the interactions in different information spaces to generate more accurate recommendation results.

（4）真实数据集上的实验结果表明，跟已有的各种算法相比，我们的混合模型在各种指标上都取得了目前最优的效果。

The experimental results on the real data set show that compared with the existing algorithms, our hybrid model has achieved the best results in various indicators.

问题定义和问题分析

场景实例？

一个真实在线服务推荐系统的仓库可以表示为,其中M是已搭建的mashup的集合，S是现有的service的集合。Mashup是由多个service构成的，它们之间存在着调用关系。我们可以把~~这种关系当做~~隐式反馈数据~~，把历史调用信息~~转化为一个mashup-service调用矩阵MS，其中第m行第s列的取值为：。每个mashup或service都存在能够描述其功能的内容信息~~，包括词序列形式的文本和离散的标签，即content=（text，tag）~~。提供商。

A real online service repositorycan be expressed as (M,S), where M is the set of mashups that have been built and S is the set of existing services. Mashup is composed of multiple service. the invocation relation can be regarded as implicit feedback data and converted into a mashup-service matrix MS, where the value at the m-th row and the s-column is:. Each mashup or service is equipped with many types of information. Content information of mashup or service (such as textual descriptions and tags) describes its function or requirements. Service has also information about its provider for use.

我们上文提到我们要解决的是session内的next-item推荐问题。具体的，在上述服务推荐系统中，在一个搭建新mashup的session中，假如用户提供了一个搭建新mashup的需求MReq（也就是新的mashup要实现的功能，也就是描述用户的功能需求的text和tag），并且在线选择了一些组件服务 context\_services，此时推荐系统怎么根据这些信息，从用户未选择的服务集合（S- context\_services）中为他推荐next-item，生成一个服务列表SL?

Suppose a developer provided his requirement to build a new mashup (in the form textual descriptions and tags, and referred to as MReq) and selected some component services (referred to as context\_services) online. At this time, how can SRS select next service to the user from the service set (S- context\_services) that the user has not selected based on this information?

问题分析：说明session的特征：关注三者交互，行为无序性，重要性不同。

分析mashup搭建过程的一些特点。

模型介绍：

**模型部分**

根据上述分析，MReq，context\_services和s之间存在着复杂的交互关系，它们影响决定了用户最终的选择行为。因此我们设计了一个深度学习模型去model their 复杂交互，预测用户选择一个候选服务的概率，。在预测对每个候选item的评分或是选择概率后，选择出评分最高的N个item作为推荐列表。

According to the above analysis, there exist interactions among MReq, context\_services and s, which determines whether the developer will select s as next service or not. Therefore, we designed a depth learning based model to model their complex interactions and predict the probability of the developer selecting s,. The top N items with the highest score or probability are selected as the recommendation list.

模型概述。我们首先分别得到三者的特征表示。然后使用MLP学习特征之间的交互。最后预测。

we obtain the feature representations of new mashup, the selected services and next service in two spaces using the content information and the invocation between mashups and services respectively. In each space, we capture their complex interaction and make a prediction by this model.

特征表示。系统中，对于三者，有很多种信息，如mashup和服务都包含内容信息，而且训练数据集中还有一些历史调用信息。基于每种信息，使用一些策略和特征提取器，我们可以得到每个mashup，已选择服务和待测服务的特征表示。然后同一个特征空间我们便可以学习它们的交互。不同特征空间的交互意义不大。每个特征空间的交互和预测就是本模型的一个实例。我们在下一节详细介绍。

需要注意的是，这里我们得到的是单个mashup和单个服务的特征表示。为了学习***MReq***，***s和context\_services***的交互，一种直观的思路是使用多个MLP分别学习 ***MReq***，***s***和每个已选择过的service的交互情况，然后再整合这些交互，但是这种方法需要消耗大量的计算资源，复杂度较高。作为替代地，我们首先使用一个向量表示***context\_services，***然后只使用一个***MLP***就可以学习***MReq***，***s和context\_services***的交互。具体地，在计算它们的整体表示时，**我们设计了一个基于attention机制的模块，它可以有效整合每个已选择service的特征表示，重点关注那些对下个service的选择行为影响比较大的services**，忽略无关service。我们在下个小节详细解释。这里我们暂时先把它简单记做：

交互层。在得到特征表示后，在这一层，我们对***MReq***，***context\_services***和***s***的交互关系建模。由于它们之间关系复杂，贸然假设和定义很可能不贴合实际情况，而DNN在理论上可以接近一切函数【】，所以我们使用一个MLP处理***MReq***，***context\_services***和***s***的功能特征的拼接结果，得到一个描述它们交互关系的低维向量。PRelu。这个过程可以描述为：

(10)

输出层。我们使用了层处理交互向量，预测调用这个服务与否的概率。在训练阶段，模型处理一个正训练样本时，在1值上的输出概率接近1；模型处理负训练样本时，在0值上的输出概率接近1。模型在Serving时，我们把模型在1值上的概率值作为当前服务的rating。

(18)

Attention块：

根据上文分析，我们在得到每个服务的表示后，需要将它们压缩为一个向量表示，去高效学习三者直接的交互。

下面讲讲我们如何整合不同的feature得到整体表示。

一种直观的方案是直接拼接所有的**。但是，拼接处理会增加特征表示的维度，增加模型的参数量，降低模型效率。另一方面，在context中选择的services之间实际上是顺序不敏感的。例如，如果开发者选中了ABC，那么无论选中的顺序是ABC还是CBA,对next-item的区别不大。但很显然，拼接的结果不满足这种性质。**

The following explains how we integrate the feature representation of each selected service to get an overall representation of context\_services.

An intuitive method is to directly concatenate the representations of all selected services. However, the concatenation will increase the dimension of final representation and reduce the efficiency of the model, especially when the developer select more and more services. On the other hand, next-item is actually not sensitive to the order of services. ~~For example, if the developer has selected three services, no matter the order of selection is ABC or CBA, there is no big difference for next-item.~~ But the concatenation result does not satisfy this property obviously.

**另外一种流行的解决方案是使用pooling方法处理每个service的表示，最终得到定长的特征表示【**Deep neural networks for youtube recommendations/Wide & deep learning for recommender systems**】。**

**其中**是已选择的第i个services的向量表示,X是已选择的services的数目。

Another popular solution is to preform average pooling or sum pooling on the representation of each service. Although getting a fixed-length feature representation, the method still has some disadvantages.

最常用的pooling方法是sum和平均池化，它们分别对已选择services的特征列表进行元素求和/平均操作。

**但是这种方法仍然存在一些问题。我们已经分析过，**下个services的选择，受到每个已选择的services的影响程度是不同的。所以整合context的特征表示时，每个已选择services的权重大小不同。然而，使用pooling方法实际上是按照同样的权重整合所有选择的services的特征表示。

已选择的多个服务所实现的功能是复杂的，所以在考虑不同候选服务时，应该关注跟这个服务相关的功能。这意味着对于不同的服务，我们得到不同的表示。

另外，在pooling方法中，用户在选择下一个services时，无论待测的service是什么，context services的特征表示是固定的，实际上这种固定维度的表示很难有效地表示出已经选择过的services中的丰富特征，比如它们已经实现的复杂功能。

显然pooling的结果是固定的，限制了它们的表达能力。

*~~用户的选择行为受相关api的影响比较大，而几乎不受无关api的影响。~~*

*~~我们只需要关注跟这个待测service相关的一些已选择services，而忽略无关的services，这样，可以更有效地表示出已选择services实现的功能。~~*

**受此启发**，~~在借助用户已选择的services信息，衡量一个待测item是否满足用户需要时，~~我们设计了一个基于attention的整合已选择services的表示的方法。**通过注意力机制，我们重点**关注那些跟待测service相关的（功能相似，可以相互替代；或者功能相异，可以互为补充）**，对选择过程影响比较大的**services~~实现的功能~~，**而忽略无关的service~~特征~~**。

整体上，我们使用各个已选择service的特征向量的加权和，去表示context services。

We use the weighted sum of feature of each selected service to represent context services.

其中w在形式上是一个已经选择的service i的权重，它的物理意义是service i与待测service s的相关程度，也可以理解为service i对用户选择s的决策行为的贡献程度 ~~影响程度，对待测service被选的~~贡献程度。

Where is formally the weight of a selected service si. The physical meaning of is the degree of correlation between si and *s* or the contribution of si to the user's selection on s.

is jointly determined by the feature of si and s. For two vectors, we measure their similarity by their element-wise multiplication and their difference by their element-wise subtraction. The results of these two operations can be seen as prior knowledge to help model the correlation between si and s. We concatenate them with the feature of si and s and input the result into an MLP to automatically learns the correlation between si and s ~~and obtains a scalar score.~~

它是由service i，service s的特征向量共同决定的。我们认为，两个特征向量的元素乘的结果，一定程度上可以衡量它们的相似度；而两个特征向量的元素减的结果，一定程度上可以衡量它们的差异（有时可以理解为功能上的互补程度）。这两个运算结果是可以帮助i和s的相关程度建模的先验知识。由于i影响选择s的行为的方式或者原因不确定，比如，可能是因为s和i之间功能互补，也可能是因为它们的功能相似，我们把这两种运算结果，与service i，service s的特征向量拼接后，一共输入到MLP中，自动学习i与s的相关程度，进而得到一个相关性score。~~所以首先我们基于一个MLP处理这两个向量以及对它们的数值操作结果。~~这个过程可以表示为：

其中， 是两个特征向量的元素乘的结果；而是两个特征向量的元素减的结果。

Where denotes element wise multiplication and denotes element-wise subtraction.

最后，我们再将这个score输入到一个softmax层中计算得到一个最终权重标量：

与pooling的方法不同，我们的方法根据已选择service和待测service的相关程度adaptively calculate the representation vector。所以在我们的方法中，不同services的对context services的特征表示的贡献权重不同，得到的context的特征表示随着待测item变化而变化。

Different from the method of pooling, we adaptively calculates the representation of selected service according to the correlation between each selected service and s. Therefore, using our method, different selected services have different contribution weights to the overall representation and the representation varies with the candidate service.

Session，online分析：

我们的模型是为在session内开发的mashup设计的。我们的模型注意到了用户以一种session的模式在线搭建mashup。它把session作为一个整体，考虑了待搭建mashup，所有已选择服务和候选服务的交互，并且突出了对选择过程影响较大的服务。

Our model notices that mashups are established and set up online in a session mode. To make recommendation, it not only models the interaction among the new mashup, all selected services and next service, but emphasizes the services that have greater impacts on the next service.

同时，当开发者选择了新服务之后，模型的已选择服务的component能够将这部分信息有效利用。模型被feed新服务的信息并提取特征之后，待测服务重新施加权重，便可重新计算三者交互，做出预测。

注意到用户在与SRS的对话中完善mashup的搭建，我们的模型可以对用户的选择行为即时做出反应。它recruit了新选择的服务，重新评价每个候选服务。re-weight每个选择的服务，再次学习它们的交互。

*不需要更新模型。*

Considering that developers gradually build new mashups in the conversation with SRS, our model respond to the developers's selection behavior instantly. It recruited the newly selected services and re-evaluated each candidate service according to updated interaction.

模型实例化

内容空间

历史信息空间

整合模型

根据上述模型框架，我们~~的模型~~在提取三者的特征向量之后，使用MLP学习它们之间的交互，最后预测在在某个搭建mashup的session中，在已选择几个的情况下，开发者选择一个候选的概率。

我们的模型输入三者的信息，学习它们的交互并做出预测。我们可以输入不同种类的信息，在不同的特征空间，从不同角度学习其交互，将其实例化。在这一节中，我们分别将内容信息和调用信息输入模型，最终得到模型的两个实例。这两个实例的主要区别在于利用信息和特征表示的不同，我们重点说明。最后，我们将这两种模型学习到的交互关系整合，得到一个混合模型。

Fed information about the new mashup, all selected services and next service, our model learns their complex interactions and makes predictions. In this section, we input two kinds of information (that is, the content information and the invocation between mashups and services) into the mode and learn the interaction in different aspects, **obtaining two instances** of our model.

We will detail the main difference between the two **instances, that is, the input** information and feature representation in this section.

Finally, we integrate the interactions learned from the two models and obtain a hybrid model.

值得说明的是，我们不像之前深度学习的常用方法一样，把对象ID作为模型的输入信息。因为在线搭建的mashup的ID，在离线训练的模型中没有出现过，也很难在线获得它的ID的高质量的embedding。

It is worth mentioning that we do not take the object ID as the input information of our model like the common recommend methods based on deep learning. The reason is that the ID of the new mashup does not appear in the model trained offline, and it is difficult to obtain its high-quality embedding with scarce training instances online.

（1）内容空间的实例

***MReq***，***context\_services***和***s***在功能上存在着复杂的关系。一方面，***s***在功能上跟***context\_services***关系复杂，可能是相互补充，或者相互替代等等。另一方面，***context\_services***和***s***的功能组合起来，共同满足用户的复杂需求***MReq***。在考虑一个候选时，用户会考虑s和MReq和context\_services之间的功能交互。所以，我们可以利用内容信息，从功能的角度，对***MReq***，***context\_services***和***s***的交互建模。

There is a complicated relationship between MReq, context services and s in function. On the one hand, s and context\_services may be substitute or complementary to each other, etc. On the other hand, context\_services and s are combined to satisfy the needs of developers, MReq. When considering a candidate, the developer will consider the functional interaction among MReq, context\_services and s. Therefore, our model employ their content information to model their interaction from a functional perspective.

服务仓库中，mashup或service的功能描述，也就是内容信息，主要可以分为两种类型：word sequence（比如描述） and separate word set（比如tags）。我们首先使用两种深度学习的方法分别处理这两种形式的内容信息，**然后将提取到的这两种特征拼接，作为mashup或服务的特征表示。**

In the service repository, the functional description of a mashup or a service after processing, that is, their content information, generally fall into two forms: word sequence (such as description) and separate word set (such as tags). So is Mreq. We first adopt two deep learning techniques to process these two forms of information respectively, and then concatenate the extracted two features as the functional representations of a mashup or a service.

First of all, we need to represent each term in the content information with a dense vector. To this end, we first convert these terms into sparse binary vectors with one-hot encoding, e.g., [0, 0, …, 1, …, 0].Then we feed the vectors into an embedding layer and map each term to a dense vector or an embedding. More specifically, the embedding layer can be viewed as a lookup table, and the embedding of a term is indeed its corresponding weights in the embedding layer.在得到每个term的embedding vector之后，我们可以将包含多个词的内容信息（word sequence 或者separate word set形式），转化为矩阵：

(1)

对于word sequence形式的内容信息，我们使用了我们在【】中提出的text\_inception来提取它的特征。受inception结构的启发，在text\_inception中，我们首先使用并行堆叠的卷积层提取词序列中的局部特征，提高了模型的效率和非线性；然后使用一个 global average pooling (GAP) layer突出有效特征，最后使用一个MLP对特征进行非线性转换。这个过程可以简写为：

对于separate word set形式的内容信息来说，多个term之间没有顺序，所以我们不能使用 用于处理词序列的text\_inception处理它。~~我们注意到，这些terms一般是用户可以理解的词汇，存在着预训练的，具有可加性的word embedding表示，例如由glove，word2vec模型训练得到的word embedding数据。因此，为了~~作为替代，我们检索并平均每个term的embedding，得到separate word set的定长的feature vector：

(8)

where is the embedding of the -th term in the set and is the size of the set.

对于一个mashup或者service，我们~~首先分别从它的word sequence and separate word set形式的内容信息中提取特征，然后拼接这两种特征，~~得到一个能够表示它的功能的特征向量：

(9)

where denotes the concatenation operation.

至此，我们可以将***MReq***，***s***和已选择的每个service的自然语言形式的内容信息表示为实值的特征向量：,，

此后，我们便可以用MLP对三者在功能上的交互建模：

最终，我们可以基于交互向量做出预测。

（2）历史信息空间的实例

除了内容信息，服务推荐系统中还存在着历史调用信息，这也是一种能够有效提高推荐效果的信息。所以，我们首先运用调用信息，得到三者的表示，然后在这个空间学习交互做出预测。这便是第二个实例。

运用历史调用信息的一种常用的框架，基于历史调用信息，离线训练设计好的模型，得到已存在的mashup m和service s在同一个空间的特征表示，最终基于这些表示做预测【是基于模型的CF和NCF等，PMF,LDA,NCF，node2vec】。

这里，我们首先将历史调用关系矩阵MS转化为图，然后使用node2vec处理它，进而可以得到仓库中存在的mashup和services（包括s）的特征表示。相比于传统的矩阵分解的方法，node2vec能够捕捉更复杂的非线性关系。

但是我们发现，这种想法在在线场景中不可行，它们只能得到训练集中已存在mashup的表示。而session内的mashup是在线搭建的，无法通过离线训练提前得到其表示，想利用session内在线选择的服务得到带搭建mashup的表示，需要更新模型~~，效果不敢保证，~~效率很低不切实际。

受User-based的方法的启发，我们首先为它从训练集中寻找一些相似的mashup（近邻mashup）。然后我们便可以使用近邻mashup的特征表示以及它们与***待搭建mashup***的相似度，加权得到***待搭建mashup***的特征表示。

具体的，给定待搭建的mashup的内容信息和已选择的服务信息，我们可以计算它与历史mashup的相似度。这里，我们为了充分利用各种信息，更精准地衡量mashup之间的相似度，使用了meta-based的相似度计算方式。【pasrec】

我们首先LDA处理文本，然后使用top3个topics作为文本的主题。然后构建了一个异构信息网：两类对象，mashup和服务。每个mashup或服务有topic表示的文本，tag，服务还有提供商信息。mashup和服务间有调用关系。我们使用六条meta-path，定量计算了两个mashup之间的6种关联，相似度（Sim i）：拥有相同的topic，相同的tag，公用一些相同的服务，公用相似的服务（拥有相同的topic，相同的tag，相同的提供商）。然后我们将每种相似度赋予不同的重要性权重，对六种相似度加权求和，得到两个mashup综合的相似度。重要性权重使用PasRec预训练好的权重参数。表示为：

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

Then We select most similar mashups of to build its neighbor mashups according to their similarities.

最后, we calculate the weighted representation of mashup using the following equation:

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

where is a neighbor mashup of , is the similarity between and , and is the representation of obtained by node2vec.

在获取context\_services的特征表示的时候，我们使用上述attention模块整合各个services的特征表示。这个过程可以简单表示为：

其中在这个特征空间中，是service 相对于s的影响权重。值得注意的是，基于历史信息得到的特征空间与基于内容信息得到的特征空间不同，所以它们的service 相对于s，在这两个空间的影响权重，**和，也是独立互不影响的。**

至此，我们得到了***MReq***，***context\_services***和***s***在同一个特征空间的表示，我们把它们拼接，使用一个MLP学习它们在这个特征空间的交互：

(17)

where denotes the representation of obtained by node2vec.

这个实例中，我们基于近邻mashup的特征表示得到***MReq***的特征表示，然后学习***MReq***与***context\_services***和***s***在这个特征空间的交互。可以理解为，借助历史信息中***NM***，***context\_services***和***s***之间的交互，推测出***MReq, context\_services***和***s***之间的交互。

在这个实例中，当开发者选择了新服务之后，除了在已选择component中加入它的信息之外，我们还可以用它来更新待搭建mashup的表示。我们需要重新计算待搭建mashup和历史mashup的相似度，然后选择最近邻并据此表示新mashup。计算相似度的方法是基于meta-path的方法，效率很高。我们可以使用一些剪枝策略，即设置一些规则以缩小候选mashup的规模，进而提高这一步的效率，例如从相同topic的mashup中选择。我们可以增量式的更新相似度。

交互模型：

上面我们已经分别基于内容信息和历史调用信息，在两个特征空间分别捕捉它们不同形式的交互。但是基于单个模型做出的预测只能利用一种信息，效果有限。下面，我们提出了一种整合多种交互的复合模型。

模型架构如图所示：

我们首先训练多个单独的模型。在每个模型中，我们输入不同的信息，然后学习三者在这个特征空间的交互。

然后，我们整合上述模型的学到的交互，做出最终更精准的预测。具体的，我们把这些模型的中间层结果，即交互向量，拼接起来，然后输入到一个MLP中，最后并作出预测。

复合模型利用了多种信息，融合多种交互，从多种角度考虑，改善推荐结果。

这也可以说明上述模型是容易扩展的。如果我们含有其他形式的新型信息，可以先在该空间学习交互并预测，然后简单地整合到原来的模型中，提升预测效果。

先预训练单个模型，然后初始化复合模型参数。最终fine-tune。

3.5模型离线学习

我们基于mashup-services调用记录搭建训练数据集，以离线训练模型。在搭建训练集时，我们随机从一个mashup的真实组件services中选择一部分作为它的context services，然后将每个未选中的组件service跟该mashup，context services组合起来，构成一个正例，标签为1。此外，我们还从非组件services中随机抽样选出一部分service，将每个service联合该mashup和context services组合成一个标签为0的负例。我们将正例集合记作，负例集合记作。

基于mashup和service的相关信息（可以组织成HIN），本模型预测一个mashup m 在选定context services的条件下调用一个candidate service s的可能性。显然，模型对正样本的预测值应当接近1，对负样本则应尽量接近0，所以模型的likelihood function是：

(19)

其中θ是模型参数，info是与mashup，services相关的所有属性信息。是模型预测的该mashup m调用service s的可能性大小。

最大化上式的似然概率等价于最小化以下目标函数：

(20)

在确定loss函数之后，我们在每个mini-batch上使用Adam算法【】训练模型。直接更新整个模型中的所有可能会导致模型收敛速度很慢，并且参数初始化也会影响深度学习模型的性能。所以我们首先训练内容交互部分，然后使用CI部分的特征提取器得到的mashup和services的内容信息的特征，训练NI部分。最后两部分分别训练完成后，再fine-tune上层的融合两部分交互向量的MLP。具体的，在内容交互部分，我们使用预训练的glove word embedding初始化mashup和service的内容信息中的每个词的word 表示，然后在训练CI的过程中根据上述目标函数更新这些embedding表示和其他参数。

然后，我们将上述 从所有近邻mashup的调用历史中获取的，表示当前的context services和s的组合关系的向量，输入到一个MLP中。这样，我们就可以学习到从可组合关系角度考虑的，context services和s的交互向量。这个过程可以表示为：

(15)

下面，我们先介绍每个部分，最后再详细介绍我们设计的用于整合单个已选择service表示的注意力模块。

更新

Session内，Item较为稳定，不用更新。User的表示，当用户动态选择的组件后，随时更新。

定期更新整个模型。扩大user库。