Experitment

## Experimental Settings

The experimental environment is a workstation with Intel Core 8 Xeon(R) @3.50 GHz, GeForce GTX 2080, and 32 GB memory. The source code implemented based on keras[[1]](#footnote-1) is available on GitHub[[2]](#footnote-2).

### Dataset and samples.

Our dataset was crawled from ProgrammableWeb, the largest online Web service registry, on July 25, 2016. We removed the mashups and services without content information and the mashups that had only one component service from the original dataset. The experimental dataset contains 1,979 mashups, 728 services, and 5802 mashup-service invocations.

The sample we used was specially designed for the online interactive scenario and consisted of a mashup to be built, one or more selected services and a service to be tested. Take a mashup, m, that actually composed of services {a,b,c,d} as an example, we randomly chose a part of its component services as selected services (e.g., {a,b}), and constructed a positive sample by combining m, {a,b}, and a random component service except a and b, e.g., c, which can be written as (m, {a,b},c). In a similar way, we can build a negative sample (m, {a,b},e), where service e is a random service not invoked by m. The size of negative samples is 12 times that of positive samples in our training sample set.

The selected services can be resampled to increase the scale of samples. For example, based on mashup m, we can choose service set {a},{b},{a,b},{c,d},{a,b,c}and {a,c,d} as the selected services and build their corresponding samples, respectively. The diversified samples imitate a real environment, help train our model better and empower it to serve developers in all circumstances. Considering that the number of a mashup’s component services seldom exceeds 4 in our dataset, we chose 1,2 and 3 component services as the selected services when building our sample set.

### Evaluation Metrics.

In this study, different approaches were evaluated based on five-fold cross-validation. We divided the 1,979 mashups into five folds, took one fold for testing and the others for training in each time. We adopted the following metrics to measure the recommendation result and averaged the metrics on five folds as the final metric.

Precision, Recall, and F1-measure at top N services in the ranking list are defined as:

|  |  |  |
| --- | --- | --- |
|  |  | (21) |
|  |  | (22) |
|  |  | (23) |

where is the mashup set appearing in the test set and denotes the size of . For mashup , is the recommended service list for it, while is its component services set.

Mean average precision (mAP) at top N services in the ranking list is defined as:

|  |  |  |
| --- | --- | --- |
|  |  | (24) |

where indicates whether a service at position in the list is an actual component service of , is the number of component services of , and denotes the number of actual component services of occurred in the top services of the ranking list.

Normalized discounted cumulative gain (NDCG) at top N services in the ranking list is defined as:

|  |  |  |
| --- | --- | --- |
|  |  | (25) |

where represents the ideal maximum DCG score that can be achieved for .

### Baseline Approaches.

In order to demonstrate the effectiveness of our model in online interaction scenarios, we selected several typical service recommendation approaches for comparison. They cover content-based methods, CF-based methods, and hybrid methods.

* WVSM: the content-based method makes a prediction of a service over a mashup using the WVSM (weighted vector space model)-based similarity between their content information.
* BPR-MF: the method uses Bayesian Personalized Ranking (BPR) to learn MF-based service recommendation models with a pairwise ranking loss.
* DHSR: the deep-learning-based approach combines collaborative filtering and content information. Specifically, in the neural CF part, the model maps mashup’s and service’s ID into their embeddings and learns their interaction with an MLP.
* SFTN: the method first calculates two probabilities that a mashup invokes a service in the next round according to their content similarity and the historical interaction between neighbor mashups and the service, and then multiplies them as a final rating according to Bayes’ theorem.
* PasRec: the method first constructs a HIN in the service recommendation scenario, then measures an overall similarity between two mashups based on the meta-paths between them, and finally adopts a user-based CF strategy to make a prediction based on the similarity. It designs a pairwise loss function and employs BPR to model optimization.
* IsRec: its framework can work well in the online interaction developing scenario. Compared with PasRec, it improves the measure of content similarity with the help of word embedding and speeds up the search for neighbor mashups by clustering existing mashups offline and classifying a new mashup online.
* DINRec: it applies Deep Interest Network in CTR to service recommendation. It exploits all available features, including the features of the mashup to be built, the selected services and the service to be tested, and learns their interaction in a well-designed network. Specifically, a local activation unit is utilized to activate the selected services related to the candidate service.

Different from others, BPR-MF and DHSR contain model-based CF modules and cannot work well in online interactive scenarios. In the first stage, they fail to work due to lack of the information about selected services. In order to apply BPR-MF and DHSR into the second stage in the experiment, we updated their models whenever developers selected new services. Moreover, we enabled DINRec to utilize the same feature extractors as ours for a fair comparison.

## Performance of MISR

As mentioned above, we introduced selected services into samples to evaluate the performance of a model in stage2, and the numbers of selected services include 1, 2, and 3. Therefore, the average value of a model's indicators in these three cases were taken as its final indicator in stage2. The performance comparison of different approaches in the first stage and the second stage is presented in table 1.

Table 1. Performance comparison of different models in stage1 and stage2.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Models | Precision@10 | | Recall@10 | | F1@10 | | NDCG@10 | | mAP@10 | |
|  | Stage1 | Stage2 | Stage1 | Stage2 | Stage1 | Stage2 | Stage1 | Stage2 | Stage1 | Stage2 |
| WVSM | 0.0484 | 0.0306 | 0.1854 | 0.1750 | 0.0743 | 0.0484 | 0.1356 | 0.1097 | 0.0853 | 0.0764 |
| BPR | - | 0.0426 | - | 0.2438 | - | 0.0675 | - | 0.1567 | - | 0.1114 |
| DHSR | - | 0.0855 | - | 0.4459 | - | 0.1400 | - | 0.3116 | - | 0.2290 |
| SFTN | 0.1198 | 0.0754 | 0.4365 | 0.4135 | 0.1815 | 0.1187 | 0.3717 | 0.2895 | 0.2707 | 0.2193 |
| IsRec | **0.1714** | 0.1004 | **0.6326** | 0.5238 | **0.2609** | 0.1558 | 0.5889 | 0.4098 | 0.4867 | 0.3359 |
| PasRec | 0.1666 | 0.1027 | 0.6205 | 0.5371 | 0.2545 | 0.1598 | **0.5909** | **0.4298** | **0.4949** | **0.3545** |
| DINRec | 0.1638 | **0.1035** | 0.6025 | **0.5395** | 0.2492 | **0.1611** | 0.5540 | 0.4172 | 0.4491 | 0.3361 |
| FI | 0.1647 | 0.1022 | 0.6093 | 0.5291 | 0.2511 | 0.1584 | 0.5636 | 0.4128 | 0.4594 | 0.3335 |
| NI | 0.1670 | 0.1002 | 0.6132 | 0.5236 | 0.2538 | 0.1559 | 0.5827 | 0.4115 | 0.4816 | 0.3343 |
| Hybrid | **0.1771** | **0.1072** | **0.6617** | **0.5572** | **0.2705** | **0.1665** | **0.6090** | **0.4391** | **0.5017** | **0.3583** |

We can find that FI performed much better than WVSM, although they both used only content information. On one hand, FI employed a CNN-based feature extractor and got high-quality and task-specific features. On the other hand, FI considered the functionality of selected services when learning the interaction between mashups and services.

BPR-MF and NI both utilized only historical invocation information, but the recommendation result of BPR-MF was much worse than that of NI, indicating that model-based CF is not applicable to interactive recommendation scenario. In order to get the embedding of a mashup built online, BPR-MF ought to update its model according to the selected services of the mashup. However, the number of the selected services was usually too small to guarantee a high-quality embedding of the mashup.

Compared with BPR-MF, SFTN also considered content information during recommendation and got a performance improvement, but its result was still poor. This was because SFTN multiplied the two probabilities obtained from content and historical information, with the assumption that they were independent of each other. But the correctness of this assumption was doubtful because we were unaware of the way how these two information affected service recommendation together.

DHSR integrated CF and content information into a network, but the performance of this deep-learning-based method was not as expected. The reasons are two-fold. Firstly, it adopted a double-tower structure to learn the interaction between a mashup to be built and a service to be tested, but did not obviously consider their interaction with the selected services. Secondly, it suffered the same problem as BPR-MF, that is, it could not get an effective representation of a mashup built online according to the limited component services that the mashup newly selected.

As a deep-learning-based method, DIN outperformed DHSR to a great extent, and achieved the best results among all baselines in some indicators. In the other indicators, PasRec and IsRec performed best among all baselines. Compared with them, our hybrid model achieved better results across all indicators in the both stages. In order to more intuitively compare the performance of our hybrid model and these optimal benchmark models (i.e., DINRec, IsRec and PasRec), table2 presents their indicators in all cases of stage2, as well as the percentage of the improvement of our model relative to these benchmarks (written as Gain in the table).

Table 2. Performance comparison of different models in all cases of stage2.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| size | Models | P@10 | Gain | R@10 | Gain | F1@10 | Gain | NDCG@10 | Gain | mAP@10 | Gain |
| 1 | IsRec | 0.1067 | 0.3% | 0.6322 | -1.4% | 0.1716 | -0.4% | 0.5246 | -2.2% | 0.4551 | -3.4% |
| PasRec | 0.1060 | 0.4% | 0.6301 | -1.0% | 0.1707 | 0.1% | 0.5241 | -2.1% | 0.4549 | -3.4% |
| DINRec | 0.1024 | 4.0% | 0.6001 | 3.9% | 0.1646 | 3.9% | 0.4854 | 5.7% | 0.4119 | 6.7% |
| Hybrid | 0.1064 | - | 0.6236 | - | 0.1710 | - | 0.5130 | - | 0.4394 | - |
| 2 | IsRec | 0.0987 | 6.8% | 0.5062 | 7.7% | 0.1526 | 7.2% | 0.3858 | 8.9% | 0.3105 | 9.0% |
| PasRec | 0.1001 | 5.3% | 0.5166 | 5.6% | 0.1553 | 5.3% | 0.4057 | 3.5% | 0.3284 | 3.0% |
| DINRec | 0.1019 | 3.4% | 0.5280 | 3.3% | 0.1584 | 3.3% | 0.4013 | 4.7% | 0.3198 | 5.8% |
| Hybrid | 0.1054 | - | 0.5453 | - | 0.1635 | - | 0.4200 | - | 0.3383 | - |
| 3 | IsRec | 0.0958 | 14.5% | 0.4330 | 16.1% | 0.1431 | 15.3% | 0.3189 | 20.5% | 0.2422 | 22.7% |
| PasRec | 0.1021 | 7.5% | 0.4645 | 8.2% | 0.1532 | 7.6% | 0.3597 | 6.8% | 0.2804 | 6.0% |
| DINRec | 0.1062 | 3.3% | 0.4902 | 2.5% | 0.1602 | 2.9% | 0.3649 | 5.3% | 0.2767 | 7.4% |
| Hybrid | 0.1097 | - | 0.5027 | - | 0.1649 | - | 0.3842 | - | 0.2973 | - |

When the number of the selected services was 2 or 3, the result of our hybrid model was the state of the art across all indicators. And as the number increased, our model got more and more advantages than these benchmarks.

DINRec was designed for e-commerce recommendation and it enforced all kinds of features to interact with each other sufficiently. However, in the service recommendation scenario, the features were extracted from content information and historical invocation respectively and in two completely independent spaces. Therefore, there was no need to combine these features across domains. By contrast, our hybrid model first learned two internal interactions in these two spaces and then fused them with an MLP, reducing the difficulty and increasing the accuracy of interaction learning.

PasRec and IsRec directly estimated a scalar score of a service over a new mashup based on the historical invocation between neighbor mashups and the service, without digging into their underlying interactions. Our model sufficiently exploited the complex interaction among the mashup, the selected services and the service by an elaborate interaction layer equipped with an MLP and an attention block.

When developers selected only one service, our hybrid model was not better than some benchmarks in some indicators. A possible reason was that the only selected service contained little useful or even disruptive information to the interaction learning.

2. Integration strategies of selected services

Our model predicts the probability of a mashup with selected services invoking a service based on their interaction. In order to learn the complex interaction more efficiently, we need to integrate the representation of each selected service and get an overall representation of all selected services in some way. An attention mechanism is utilized in our model to accomplish this task. To compare the impact of different integration strategies on the model's recommendation result, we adopt three other strategies to replace the attention mechanism, and proposed three model variants.

MISR-average: it preformed average pooling on the representation of each service and assigned equal weight to them.

MISR-concate: it directly concatenated the representation of each service. We truncated or padded the selected service set to get a final representation with a fixed size.

MISR-None: it discarded the selected services and did not consider them when learning the complex interaction.

Based on the content information or historical information, we evaluated the performances of different model variants in all cases of stage2. The indicators and the percentage of the improvement of our model relative to these variants were shown in table 3.

Table 3. Performance comparison of different variants of MISR

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | FI | | | | NI | | | |
| Mode | Size | F1@10 | Gain | NDCG@10 | Gain | F1@10 | Gain | NDCG@10 | Gain |
| MISR | 1 | 0.1644 | - | 0.4884 | - | 0.1621 | - | 0.4819 | - |
| 2 | 0.1563 | - | 0.3923 | - | 0.1543 | - | 0.3913 | - |
| 3 | 0.1543 | - | 0.3576 | - | 0.1512 | - | 0.3614 | - |
| MISR-None | 1 | 0.1615 | 1.8% | 0.4784 | 2.1% | 0.1618 | 0.1% | 0.4784 | 0.7% |
| 2 | 0.1462 | 6.9% | 0.3646 | 7.6% | 0.1485 | 3.9% | 0.3698 | 2.5% |
| 3 | 0.1402 | 10.1% | 0.3093 | 15.6% | 0.1427 | 6.0% | 0.3318 | 8.9% |
| MISR-Concate | 1 | 0.1626 | 1.1% | 0.4847 | 0.8% | 0.1607 | 0.9% | 0.4742 | 1.6% |
| 2 | 0.1517 | 3.0% | 0.3806 | 3.1% | 0.1475 | 4.6% | 0.3698 | 5.8% |
| 3 | 0.1487 | 3.8% | 0.3432 | 4.2% | 0.1443 | 4.8% | 0.3437 | 5.2% |
| MISR-Average | 1 | 0.1634 | 0.6% | 0.4821 | 1.3% | 0.1616 | 0.3% | 0.4763 | 1.2% |
| 2 | 0.1522 | 2.7% | 0.3833 | 2.4% | 0.1519 | 1.6% | 0.3857 | 1.4% |
| 3 | 0.1481 | 4.2% | 0.3388 | 5.5% | 0.1470 | 2.9% | 0.3493 | 3.5% |

The performance of MISR-None was mostly the worst, indicating that selected services did play an important role in online interactive recommendation and we should make the best of them. When developers selected only one service, there was no obvious difference among the performances of MISR, MISR-concate and MISR-average. When developers selected two and more service, our attention-based model performed better than MISR-average and MISR-concate. This was because it paid more attention to the selected services more related to current prediction, assigned distinct weight to each selected service, and obtained an adaptable representation specific to different service to be tested.

We can also find that the improvement of attention mechanism relative to average pooling and concatenation was not as prominent as expected. A possible reason was that the number of selected services in our experiments was small, i.e., not bigger than 3. This could be verified by the observation that the percentage of the improvement gradually increased as the number of selected services increased.

3. Impact of the Size of Neighbor mashups.

In the neighbor interaction module, we infer the interaction between a new mashup and a service based on those among neighbor mashups and the service. This subsection will seek for an optimal value for neighbor size,, that helps the module achieve best result. The candidate value was set from 10 to 50 by step 10.

Table 3. Recommendation Performance with Different K

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| K | P@10 | R@10 | F1@10 | NDCG@10 | mAP@10 |
| 10 | 0.0981 | 0.5433 | 0.1552 | 0.4340 | 0.3607 |
| 20 | **0.1005** | **0.5542** | **0.1587** | **0.4434** | **0.3686** |
| 30 | 0.0982 | 0.5398 | 0.1549 | 0.4348 | 0.3621 |
| 40 | 0.0977 | 0.5357 | 0.1539 | 0.4290 | 0.3561 |
| 50 | 0.0968 | 0.5341 | 0.1528 | 0.4264 | 0.3539 |

As shown in Table 3, the indicators increased as the value of K increased from 10 to 20. The reason was that the module could absorb and exploit more beneficial information from neighbor interactions. However, the opposite was true when K exceeded 20. Perhaps the introduction of noisy data brought negative interference to interaction learning in this phase. Therefore, we set to 20 in our experiments.

1. https://keras.io [↑](#footnote-ref-1)
2. https://github.com/ssea-lab/MISR [↑](#footnote-ref-2)