Technical Report for KDD CUP TASK3

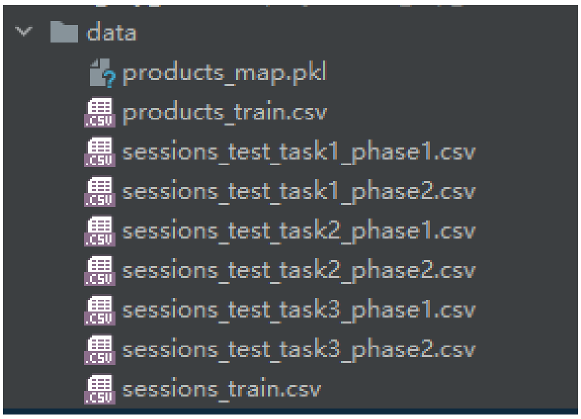
Main idea

We recommend the next item based on a simple co-visiting graph (the number of times two items co-occur in the same session). But the difference in task3 is that if the item prediction is wrong, it does not mean that the BLEU score is low, because the title may still be very similar compared to ground true. In order to avoid the risk of dissimilarity between the title and the ground true due to incorrect product predictions, we do not directly take the title of the predicted item, but take the intersection of its token and the title of the last item (the titles of two adjacent item are often similar, and we have tried that if the title of the last item is used directly as the predicted title, the BLEU score is also considerable).

Code run

1. Download data from https://drive.google.com/drive/folders/1oKf7yaeo3AxtxWAyaRoHIa5ESmrlWLed?usp=sharing. Note that we are not using any external data.

All data looks like the following,



Run our code:

```
python prediction.py
```

Details

1. Read all items, training set, test set, and the mapping dictionary from item id to title.

```python
products = read_product_data()  # load products
hist_data = read_train_data()  # load training data
test_sessions = read_test_data(task)  # load test data
test_locale_names = test_sessions['locale'].unique()
with open("data/products_map.pkl", "rb") as tf:  # load id:title dictionary
    products_map = pickle.load(tf)
```

2. Construct co-visiting diagram. Here we are based on two considerations. First, only the most recent five items are used for construction per session. Because earlier interactions may have lost their timeliness. Second, among the five items, not any two co-visiting item have the same weight. If the distance between two items (judged by the order of interactions) is greater, the weight will be smaller, so that we can focus more on adjacent interacting items.

   In addition, in addition to the construction based on the training set, we can also use the test set of the phase 1 of task 3, and the test sets of task1 and task2 for constructing. For the training set, we use the 4 items of history and the next item. For the other supplementary data, we select five historical commodities due to missing labels.

```python
# construct co-visiting graph
all_id = products['id'].unique()
id_to_idx = {id: i for i, id in enumerate(all_id)}
idx_to_id = {i: id for i, id in enumerate(all_id)}
graph = {id: {} for i, id in enumerate(all_id)}
phase1 = pd.read_csv('data/sessions_test_task3_phase1.csv')
phase2 = pd.read_csv('data/append_data.csv')
for _, row in hist_data.iterrows():  # use training data to construct co-visiting graph
    items = ([s.strip("'\n") for s in row['prev_items'][1:-1].split(" ")] + [row['next_item'].strip()])[::-1][:5]

    for i in range(0, len(items)):
        for j in range(i + 1, len(items)):
            if (items[j] != items[i]):
                try:
                    graph[items[i]][items[j]] += 1.1 - abs(j - i) * 0.2
                except:
                    graph[items[i]][items[j]] = 1.1 - abs(j - i) * 0.2
                try:
                    graph[items[j]][items[i]] += 1.1 - abs(j - i) * 0.2
                except:
                    graph[items[j]][items[i]] = 1.1 - abs(j - i) * 0.2
for _, row in phase1.iterrows():  # use task3 phase1 data to construct co-visiting graph
    items = ([s.strip("'\n") for s in row['prev_items'][1:-1].split(" ")])[::-1][:5]
    for i in range(0, len(items)):
        for j in range(i + 1, len(items)):
            if (items[j] != items[i]):
                try:
                    graph[items[i]][items[j]] += 1.1 - abs(j - i) * 0.2
                except:
                    graph[items[i]][items[j]] = 1.1 - abs(j - i) * 0.2
                try:
                    graph[items[j]][items[i]] += 1.1 - abs(j - i) * 0.2
                except:
                    graph[items[j]][items[i]] = 1.1 - abs(j - i) * 0.2
with open("data/graph_session_last_copy.pkl", "wb") as tf:
    pickle.dump(graph, tf)
for _, row in phase2.iterrows():  # use task1& task2 data to construct co-visiting graph
    items = ([s.strip("'\n") for s in row['prev_items'][1:-1].split(" ")])[::-1][:5]
    for i in range(0, len(items)):
        for j in range(i + 1, len(items)):
            if (items[j] != items[i]):
                try:
                    graph[items[i]][items[j]] += 1.1 - abs(j - i) * 0.2
                except:
                    graph[items[i]][items[j]] = 1.1 - abs(j - i) * 0.2
                try:
                    graph[items[j]][items[i]] += 1.1 - abs(j - i) * 0.2
                except:
                    graph[items[j]][items[i]] = 1.1 - abs(j - i) * 0.2
with open("data/graph_session_last.pkl", "wb") as tf:
    pickle.dump(graph, tf)
with open("data/graph_session_last_copy.pkl", "rb") as tf:
    graph = pickle.load(tf)
with open("data/graph_session_last.pkl", "rb") as tf:
    graph1 = pickle.load(tf)
```

3. For each session, we choose the last item in history (Line 127-135 in utils.py). Based on the co-visiting graph, we can get the candidate items that appear together with the item. If the frequency of a certain product is particularly prominent (here we are based on the 6-sigma principle, Line 145), we select it as the final candiate item, otherwise, there is no candidate.

```python
    # obtain the latest record of the corresponding language in the history record, and we predict the next title based on this record
    for i in range(len(his_list)):
        try:
            title = products[locale][his_list[i]]
            if (isinstance(title, str)):
                his_titles.append(title)
                his_titles_id.append(his_list[i])
        except:
            pass

    # we only use the last interacted item
    his_titles = his_titles[:1]

    # get the last item's co-visiting items
    titles_map = graph[his_titles_id[0]]
    titles_map = sorted(titles_map.items(), key=lambda x: x[1], reverse=True)
    titles_map = {v[0]: v[1] for v in titles_map}
    titles_array = np.array(list(titles_map.values()))
    threshold = np.mean(titles_array) + 6. * np.std(titles_array)

    candidate = []
    for key, value in titles_map.items():
        if (
                value > threshold):  # if the maximum item's weight in co-visiting items is larger than a certain threshold (here is 6-sigma as shwon in Line 145)
            try:
                candidate.append(" ".join(products[locale][key].split()).replace("  ", " "))
                break
            except:
                pass
        else:
            break
```

4. Since the recommendation strategy is simple, if there is no candidate item that is particularly prominent, to avoid recommendation errors, we directly select the last interacted item’s title.

```python
    if (len(candidate) == 0):
        answer = his_titles[0]
```

5. If there is a final candidate, we still will not choose it directly, because we cannot bear the risk of it being wrong and causing a sharp drop in BLEU. Therefore, we take the intersection of the candidate title and the title of the last interacted item. If the number of tokens they overlap is greater than xx% tokens of the candidate, we will choose this intersection as the final predicted title. If there are too few intersections, in order to avoid risks, we still choose the title of the last interacted product as the predicted title.

```python
else:
    list1 = split_notation(his_titles[0].lower().split())
    set1 = set(list1)
    list2 = split_notation(candidate[0].lower().split())
    list22 = split_notation(candidate[0].split())
    set2 = set(list2)
    set3 = set1 & set2
    list3 = []
    list4 = []
    for iii in range(len(list2)):
        if (list2[iii] in set3):
            list3.append(list22[iii])
            if (iii > 0):
                if ("%s %s" % (list22[iii - 1], list22[iii]) in candidate[0]):
                    if (len(list4) > 0 and list4[-1] != " "):
                        list4.append(" ")
                    list4.append(list22[iii])
                elif ("%s%s" % (list22[iii - 1], list22[iii]) in candidate[0]):
                    list4.append(list22[iii])
            else:
                list4.append(list22[iii])
        else:
            if (iii > 0):
                if ("%s %s" % (list22[iii - 1], list22[iii]) in candidate[0] and len(list4) > 0 and
                        list4[
                            -1] != " "):
                    list4.append(" ")
    if (locale == 'JP' and len("".join(list3)) > len("".join(list2)) * 0.8):
        list4 = finetune(list4)
        answer = "".join(list4)
        candidate_count += 1
    elif (locale != 'JP' and len(list3) > len(list2) * 0.8):
        list4 = finetune(list4)
        answer = "".join(list4)
        candidate_count += 1
    else:
        answer = his_titles[0]
```

6. The final evaluation score of the method on the test set is 0.27130.

233280	ustc-gobble	graded	0.27130	0.99993	Graded successfully!	Sun, 4 Jun 2023 14:54:51	View