Helper Scripts

Explanation of the scripts that is used for various purposes

Utils.py

Utils.py is responsible for helping front end related jobs. It has methods to create dictionaries that is used for showing recommendations on the web page.

```
def create_recommendations(similar_items):
    """ param: similar_items -> dict """
```

Summary

Gets the dictionary for similar items and extracts the necessary information for the frontend.

Workflow

- Retrieve all the items from the Django Model
- Start iterating through similar items dictionary:
    - Create a single item object and iterates its attribute-value.
    - Save the attributes that we would like to show them in frontend
- At the end, add URL of the item into the dictionary
- Return the dictionary back.

Return

Item dictionary that holds item information

```
def create_item_dict(context, create_item_detail=False):
    """
    param: context: Contet of the current web page
    param: create_item_detail:
                If true, create a dictionary for an individual item to use in
Item DetailView.
                If false, create a dictionary that contains 6 items for each
page on the homepage.
    """
```

Summary

Creates a dictionary that contains information of a specific item or items, depending on if user is in the homepage or dedicated item page.

Workflow

- Retrieve all the items from the Django Model
    - If user clicked on an item (create_item_details = True)
        - Query the item in database.
        - Loop through its attributes-values and concatenate them with **#** symbol.

        - Return a dictionary that contains id, attribute and value of the item

    - If user is in homepage (create_item_details = False)
        - Since, we are in the homepage now, we get the context of the page which has 'paginate_by' number of items
        - Get item information from context. -> context['item_list']
        - Loop through its attributes-values and concatenate them with **#** symbol.
        - Return a dictionary that contains id, attribute and value of 'paginated_by' number of items

Return

This method calls 'process_dictionary' method, that is described below, with prepared dictionary.

```
def process_dictionary(item_dictionary):
    """
    param: param item_dictionary: dictionary that holds item(s) data which is
concatenated by '#'        symbol.
                    { 'item_id': { 'att1#att2#att3':'val1#val2#val3# } }
    """
```

Summary

Receives the dictionary and transformers it in a way to be used in frontend.

Workflow

- Iterate through the item_dictionary:
    - Create an item object and split the attributes. (Remember that they were concatenated by using **#** symbol)
    - Iterate through the attribute and value lists together and create the final dictionary with the structure that is described on **return section**

Return

Dictionary that holds item information with better structure, so it can be processed on frontend easily.

```
    { 'item_id': { 'att1':'val1', 'att2':'val2', 'att3':'val3' } }
```

DatabaseOperations.py

Travel Agency Project is currently using database operations to get information about:

- Users
- Items
- Recommendation Models

Below you can find the code explanations of each section.

User Operations

- add_new_user
- create_user_sequences
- get_all_user_sequences
- get_user_sequences(item_id)

Add New User

```
def add_new_user(_user_id, item_id):
```

Summary

parameter: _user_id: Current user Id.

parameter: _user_id: Current item user clicked on.

Workflow

- Query the database and check if user is present.
    - If id is already in database, add a new row with user id and item id.
    - If id is not present in the database, add a new row with user id and item id.

  TODO: Do we really need to check if user is in database since either way we are adding the same row.

Return

New user and item (s)he clicked on is added to the database.

Create User Sequences

```
def create_user_sequences():
```

Summary

Method iterates through every row in the ecommerce_user table. Find the same users and items that are clicked by the user. Merges all items in one list to create a sequence.

PK	user_Id	user_sequence
1	531244	[4805,3,2155,4728]
2	531245	[1881,1923,9999]

Workflow

- Get all user ids from database.
- Iterate through user list and query for all items that user visited before.
- If user has more than 2 item clicked:
    - Add those items to list by using ',' separator.
    - Insert the sequence to database so we can use it later.

(Do nothing if user has less than 2 item in the sequence.)

Return

User sequences are created and saved in DB.

Get All User Sequences

```
def get_all_user_sequences():
```

Summary

Method is used to retrieve sequence data to train item2vec.

Workflow

- Get all user sequences from database.
- Iterate through the sequences and tokenize each item id.

Return

List of user sequences: [user_id, ['item_id1', 'item_id2', 'item_id3']]

Get User Sequences by given ID

```
def retrieve_user_sequences(item_id):
```

Summary

Get all user sequences and find the ones with current item id in it.

Method not only finds user sequences from the database but also trims the user sequences based on the given item id.

As an example:

Imagine we have following user sequence: **[1881, 25, 1923, 97, 1938]**

And user clicked on item '1923'. That means next item will be '97'.

So I am trimming the sequence after item '1923'. Then new sequence will look like: **[97, 1938]**

Then I can compare my recommendations with this new sequence since I do not think recommending an item that user has already visited is not a success.

Workflow

- Get user sequences which has item_id in it. (SQL **LIKE**)
- Create an empty dictionary to keep user sequences and iterate through the user sequences.
    - Tokenize the item ids since we need string representation of the item ids.
    - Locate the position of given item id in the list and remove everything before the item (including the item itself)
- Check new user sequence list for the sequences longer than 2 item and add those to the dictionary.

Return

user_sequences_dict

Item Operations

- add_new_item
- delete_item
- update_item
- find_item_id
- daily_updates

Add New Item

```
def update_item(conn, _object_id, _attribute, _value):
```

Summary

conn: DB connection object

_object_id: Item id

_attribute: Item attribute

_value: Attribute value

Adds item to the database.

Return

New item with attribute - value is added to the database.

Delete Item

```
def delete_item(conn, _object_id):
```

Summary

conn: DB connection object

_object_id: ID of the object to be deleted

Return

Deletes item from the database.

Update Item

```
def update_item(conn, _object_id, _attribute, _value):
```

Summary

conn: DB connection object

_object_id: ID of the object to be updated

_attribute: Attribute to be updated (must exist)

_value: New value for the attribute

Updates an item.

Return

Item attribute is updated with a new value

Daily Updates

```
def daily_updates(update_dict):
    """
        Structure for daily updates :
            {
            'new_item':
                      { 'attribute_1' : 'value_1',
                        'attribute_2' : 'value_2', },
            'update_item':
                      { 'attribute_1' : 'value_1',
                        'attribute_2' : 'value_2', },
            'delete_item':
                      [ 'item_1_id', 'item_2_id' ]
            }
    """
```

Summary

Receives an update batch and performs 'adding new item', 'updating item' and 'deleting item' operations.

update_dict: Dictionary which its structure is shown above.

Recommendation Operations

- save_model
- load_model
- change_active_model
- delete_old_models

Save Model

```
def save_model(identifier, name, model):
```

Summary

identifier: Recommender tag (e.g. 'i2v' , 'd2v')

name: name of the model which is described in the model code

model: binary representation of the model

Adds the recommendation model into DB.

Return

New model is added to the database.

Load Model

```
def load_model(identifier, name=None):
```

Summary

If model name is not given, the latest model will be loaded.

identifier: Recommender tag (e.g. 'i2v' , 'd2v')

name: name of the model which is described in the model code

Return

Loades the model from the database.

Change Active Model

```
def change_active_model(model_name):
```

Summary

Find the active model and change its state to false. Then find the given model an activate it.

model_name: model name to be activated

Return

Current recommender is changed with the given one.

Delete Old Models

```
def delete_old_models():
```

Summary

Deletes the saved models older than 30 days.

PreProcess.py

PreProcess.py is heavily use for data manipulation. It has methods for RNN to create user sequences, convert them to GRU4REC format and also other methods for text cleaning.

```
def create_sequence_ids(df):
```

Summary

Aggregate the items that user clicked on in decided 'sequence_interval'

Workflow

- Group the dataframe by using user_id and timestamp columns
- Create user sequences:
    - Iterate through the dataframe:
        - Since the dataframe is ordered, check if the next item user clicked is in 30 days interval.
        - If it is, consider it as in the same session.
        - If not, increment session number.
    - At the end, we will have session numbers and their corresponding items
    - Then convert date to Unix Timestamp for RNN

Return

Dataframe that is modified.

```
def dataset_to_gru4rec_format(dataset):
```

Summary

Convert a list of sequences to GRU4Rec format.

Workflow

- Group the sequences based on ascending timestamp.
-

Return

'Unstacked' Dataframe.

```
def clean_text(df):
```

Summary

Cleans the texts inside the given Dataframe.

Workflow

- Iterate through the dataframe:
    - Lowercase the letters
    - Remove the stop words. (Except the comma)

Return

Cleaned Dataframe

Metrics.py

Metrics.py is used to evaluate performance of the recommendations. Currently, it has methods to calculate **Precision, Recall,** and **MMR**.

```
def precision(ground_truth, prediction):
    """
    Compute Precision metric
    :param ground_truth: the ground truth set or sequence
    :param prediction: the predicted set or sequence
    :return: the value of the metric
    """
```

```
def recall(ground_truth, prediction):
    """
    Compute Recall metric
    :param ground_truth: the ground truth set or sequence
    :param prediction: the predicted set or sequence
    :return: the value of the metric
    """
```

```
def mrr(ground_truth, prediction):
    """
    Compute Mean Reciprocal Rank metric. Reciprocal Rank is set 0 if no
predicted item is in             contained the ground truth.
    :param ground_truth: the ground truth set or sequence
    :param prediction: the predicted set or sequence
    :return: the value of the metric
    """
```