Algorithm/DeepFM/main.py

```
if __name__ == '__main__':
    import Recommender_System.utility.gpu_memory_growth
    from Recommender_System.data import data_loader, data_process
    from Recommender_System.algorithm.DeepFM.model import DeepFM_model
    from Recommender_System.algorithm.train import train

    n_user, n_item, train_data, test_data, topk_data = data_process.pack(data_loader.ml100k)

    model = DeepFM_model(n_user, n_item, dim=8, layers=[16, 16, 16], l2=1e-5)

    train(model, train_data, test_data, topk_data, epochs=10)
```

algorithm/DeepFM/model.py

```
import tensorflow as tf
from Recommender_System.utility.decorator import logger


@logger('初始化DeepFM模型：', ('n_user', 'n_item', 'dim', 'layers', 'l2'))
def DeepFM_model(n_user: int, n_item: int, dim=8, layers=[16, 16, 16], l2=1e-6) -> tf.keras.Model:
    l2 = tf.keras.regularizers.l2(l2)

    user_id = tf.keras.Input(shape=(), name='user_id', dtype=tf.int32)
    user_embedding = tf.keras.layers.Embedding(n_user, dim, embeddings_regularizer=l2)(user_id)

    item_id = tf.keras.Input(shape=(), name='item_id', dtype=tf.int32)
    item_embedding = tf.keras.layers.Embedding(n_item, dim, embeddings_regularizer=l2)(item_id)

    user_bias = tf.keras.layers.Embedding(n_user, 1, embeddings_initializer='zeros')(user_id)
    item_bias = tf.keras.layers.Embedding(n_item, 1, embeddings_initializer='zeros')(item_id)
    fm = tf.reduce_sum(user_embedding * item_embedding, axis=1, keepdims=True) + user_bias +
item_bias

    deep = tf.concat([user_embedding, item_embedding], axis=1)
    for layer in layers:
        deep = tf.keras.layers.Dense(layer, activation='relu', kernel_regularizer=l2)(deep)
    deep = tf.keras.layers.Dense(1, kernel_regularizer=l2)(deep)

    out = tf.keras.activations.sigmoid(fm + deep)
```

```
    return tf.keras.Model(inputs=[user_id, item_id], outputs=out)


if __name__ == '__main__':
    tf.keras.utils.plot_model(DeepFM_model(1, 1), 'graph.png', show_shapes=True)
```

algorithm/FM/main.py

```
if __name__ == '__main__':
    import Recommender_System.utility.gpu_memory_growth
    from Recommender_System.data import data_loader, data_process
    from Recommender_System.algorithm.FM.model import FM_model
    from Recommender_System.algorithm.train import train

    n_user, n_item, train_data, test_data, topk_data = data_process.pack(data_loader.ml100k)

    model = FM_model(n_user, n_item, dim=16, l2=1e-6)

    train(model, train_data, test_data, topk_data, epochs=10, batch=512)
```

algorithm/FM/model.py

```
import tensorflow as tf
from Recommender_System.utility.decorator import logger


@logger('初始化FM模型：', ('n_user', 'n_item', 'dim', 'l2'))
def FM_model(n_user: int, n_item: int, dim=8, l2=1e-6) -> tf.keras.Model:
    l2 = tf.keras.regularizers.l2(l2)

    user_id = tf.keras.Input(shape=(), name='user_id', dtype=tf.int32)
    user_embedding = tf.keras.layers.Embedding(n_user, dim, embeddings_regularizer=l2)(user_id)
    user_bias = tf.keras.layers.Embedding(n_user, 1, embeddings_initializer='zeros')(user_id)

    item_id = tf.keras.Input(shape=(), name='item_id', dtype=tf.int32)
    item_embedding = tf.keras.layers.Embedding(n_item, dim, embeddings_regularizer=l2)(item_id)
    item_bias = tf.keras.layers.Embedding(n_item, 1, embeddings_initializer='zeros')(item_id)

    x = tf.reduce_sum(user_embedding * item_embedding, axis=1, keepdims=True) + user_bias +
item_bias
    out = tf.keras.activations.sigmoid(x)
```

```
    return tf.keras.Model(inputs=[user_id, item_id], outputs=out)



if __name__ == '__main__':
    tf.keras.utils.plot_model(FM_model(1, 1), 'graph.png', show_shapes=True)
```

algorithm/KGCN/layer.py

```
from abc import abstractmethod
import tensorflow as tf



class Aggregator(tf.keras.layers.Layer):
    def __init__(self, activation='relu', kernel_regularizer=None, **kwargs):
        super(Aggregator, self).__init__(**kwargs)
        self.activation = tf.keras.activations.get(activation)
        self.kernel_regularizer = tf.keras.regularizers.get(kernel_regularizer)

    def call(self, inputs, **kwargs):
        self_vectors, neighbor_vectors, neighbor_relations, user_embeddings = inputs

        _, neighbor_iter, dim = self_vectors.shape
        neighbor_size = kwargs['neighbor_size']

        neighbor_vectors = tf.reshape(neighbor_vectors, shape=(-1, neighbor_iter, neighbor_size, dim))
        neighbor_relations = tf.reshape(neighbor_relations, shape=(-1, neighbor_iter, neighbor_size, dim))

        outputs = self._call(self_vectors, neighbor_vectors, neighbor_relations, user_embeddings,
**kwargs)
        if self.activation is not None:
            outputs = self.activation(outputs)
        return outputs

    @abstractmethod
    def _call(self, self_vectors, neighbor_vectors, neighbor_relations, user_embeddings, **kwargs):
        # self_vectors: [batch, neighbor_iter, dim]
        # neighbor_vectors: [batch, neighbor_iter, neighbor_size, dim]
        # neighbor_relations: [batch, neighbor_iter, neighbor_size, dim]
        # user_embeddings: [batch, dim]
        pass
```

```
    def _mix_neighbor_vectors(self, neighbor_vectors, neighbor_relations, user_embeddings):
        dim = user_embeddings.shape[-1]
        avg = False
        if not avg:
            user_embeddings = tf.reshape(user_embeddings, shape=(-1, 1, 1, dim))  # [batch, 1, 1, dim]

            user_relation_scores = tf.reduce_mean(user_embeddings * neighbor_relations, axis=-1)  # [batch, neighbor_iter, neighbor_size]
            user_relation_scores_normalized = tf.nn.softmax(user_relation_scores, axis=-1)  # [batch, neighbor_iter, neighbor_size]
            user_relation_scores_normalized = tf.expand_dims(user_relation_scores_normalized, axis=-1)  # [batch, neighbor_iter, neighbor_size, 1]

            neighbors_aggregated = tf.reduce_mean(user_relation_scores_normalized * neighbor_vectors, axis=2)  # [batch, neighbor_iter, dim]
        else:
            neighbors_aggregated = tf.reduce_mean(neighbor_vectors, axis=2)  # [batch, neighbor_iter, dim]

        return neighbors_aggregated


class SumAggregator(Aggregator):
    def build(self, input_shape):
        dim = input_shape[-1][-1]
        self.kernel = self.add_weight('kernel', shape=(dim, dim), initializer='glorot_uniform', regularizer=self.kernel_regularizer)
        self.bias = self.add_weight('bias', shape=(dim,), initializer='zeros')

    def _call(self, self_vectors, neighbor_vectors, neighbor_relations, user_embeddings, **kwargs):
        _, neighbor_iter, dim = self_vectors.shape
        neighbors_agg = self._mix_neighbor_vectors(neighbor_vectors, neighbor_relations, user_embeddings)  # [batch, neighbor_iter, dim]

        output = tf.reshape(self_vectors + neighbors_agg, shape=(-1, dim))  # [batch * neighbor_iter, dim]
        #if kwargs['training']:
        #    output = tf.nn.dropout(output, rate=0.2)
        output = tf.nn.bias_add(tf.matmul(output, self.kernel), self.bias)  # [batch * neighbor_iter, dim]

        return tf.reshape(output, shape=(-1, neighbor_iter, dim))  # [batch, neighbor_iter, dim]
```

```
class ConcatAggregator(Aggregator):

    def build(self, input_shape):

        dim = input_shape[-1][-1]

        self.kernel = self.add_weight('kernel', shape=(dim * 2, dim), initializer='glorot_uniform',
regularizer=self.kernel_regularizer)

        self.bias = self.add_weight('bias', shape=(dim,), initializer='zeros')


    def _call(self, self_vectors, neighbor_vectors, neighbor_relations, user_embeddings, **kwargs):

        _, neighbor_iter, dim = self_vectors.shape

        neighbors_agg = self._mix_neighbor_vectors(neighbor_vectors, neighbor_relations,
user_embeddings)  # [batch, neighbor_iter, dim]


        output = tf.concat([self_vectors, neighbors_agg], axis=2)  # [batch, neighbor_iter, dim * 2]

        output = tf.reshape(output, shape=(-1, dim * 2))  # [batch * neighbor_iter, dim * 2]

        #if kwargs['training']:

        #    output = tf.nn.dropout(output, rate=0.2)

        output = tf.nn.bias_add(tf.matmul(output, self.kernel), self.bias)  # [batch * neighbor_iter, dim]


        return tf.reshape(output, shape=(-1, neighbor_iter, dim))  # [batch, neighbor_iter, dim]



class NeighborAggregator(Aggregator):

    def build(self, input_shape):

        dim = input_shape[-1][-1]

        self.kernel = self.add_weight('kernel', shape=(dim, dim), initializer='glorot_uniform',
regularizer=self.kernel_regularizer)

        self.bias = self.add_weight('bias', shape=(dim,), initializer='zeros')


    def _call(self, self_vectors, neighbor_vectors, neighbor_relations, user_embeddings, **kwargs):

        _, neighbor_iter, dim = self_vectors.shape

        neighbors_agg = self._mix_neighbor_vectors(neighbor_vectors, neighbor_relations,
user_embeddings)  # [batch, neighbor_iter, dim]


        output = tf.reshape(neighbors_agg, shape=(-1, dim))  # [batch * neighbor_iter, dim]

        #if kwargs['training']:

        #    output = tf.nn.dropout(output, rate=0.2)

        output = tf.nn.bias_add(tf.matmul(output, self.kernel), self.bias)  # [batch * neighbor_iter, dim]
```

```
        return tf.reshape(output, shape=(-1, neighbor_iter, dim))  # [batch, neighbor_iter, dim]
```

algorithm/KGCN/main.py

```
if __name__ == '__main__':
    import Recommender_System.utility.gpu_memory_growth
    from Recommender_System.algorithm.KGCN.tool import construct_undirected_kg, get_adj_list
    from Recommender_System.algorithm.KGCN.model import KGCN_model
    from Recommender_System.algorithm.KGCN.train import train
    from Recommender_System.data import kg_loader, data_process
    import tensorflow as tf

    n_user, n_item, n_entity, n_relation, train_data, test_data, kg, topk_data =
data_process.pack_kg(kg_loader.ml1m_kg1m, negative_sample_threshold=4)

    neighbor_size = 16
    adj_entity, adj_relation = get_adj_list(construct_undirected_kg(kg), n_entity, neighbor_size)

    model = KGCN_model(n_user, n_entity, n_relation, adj_entity, adj_relation, neighbor_size,
iter_size=1, dim=16, l2=1e-7, aggregator='sum')

    train(model, train_data, test_data, topk_data, optimizer=tf.keras.optimizers.Adam(0.01), epochs=10,
batch=512)
```

algorithm/KGCN/model.py

```
if __name__ == '__main__':
    import Recommender_System.utility.gpu_memory_growth
    from Recommender_System.algorithm.KGCN.tool import construct_undirected_kg, get_adj_list
    from Recommender_System.algorithm.KGCN.model import KGCN_model
    from Recommender_System.algorithm.KGCN.train import train
    from Recommender_System.data import kg_loader, data_process
    import tensorflow as tf

    n_user, n_item, n_entity, n_relation, train_data, test_data, kg, topk_data =
data_process.pack_kg(kg_loader.ml1m_kg1m, negative_sample_threshold=4)

    neighbor_size = 16
    adj_entity, adj_relation = get_adj_list(construct_undirected_kg(kg), n_entity, neighbor_size)

    model = KGCN_model(n_user, n_entity, n_relation, adj_entity, adj_relation, neighbor_size,
iter_size=1, dim=16, l2=1e-7, aggregator='sum')
```

train(model, train_data, test_data, topk_data, optimizer=tf.keras.optimizers.Adam(0.01), epochs=10, batch=512)

algorithm/KGCN/train.py

```
import time
from typing import List, Tuple
import tensorflow as tf
from Recommender_System.utility.decorator import logger
from Recommender_System.utility.evaluation import TopkData
from Recommender_System.algorithm.train import prepare_ds, get_score_fn
from Recommender_System.algorithm.common import log, topk


@logger('开始训练．', ('epochs', 'batch'))
def train(model: tf.keras.Model, train_data: List[Tuple[int, int, int]], test_data: List[Tuple[int, int, int]],
          topk_data: TopkData = None, optimizer=None, epochs=100, batch=512):
    if optimizer is None:
        optimizer = tf.keras.optimizers.Adam()

    train_ds, test_ds = prepare_ds(train_data, test_data, batch)

    loss_mean_metric = tf.keras.metrics.Mean()
    auc_metric = tf.keras.metrics.AUC()
    precision_metric = tf.keras.metrics.Precision()
    recall_metric = tf.keras.metrics.Recall()
    loss_object = tf.keras.losses.BinaryCrossentropy()
    if topk_data:
        score_fn = get_score_fn(model)

    def reset_metrics():
        for metric in [loss_mean_metric, auc_metric, precision_metric, recall_metric]:
            tf.py_function(metric.reset_states, [], [])

    def update_metrics(loss, label, score):
        loss_mean_metric.update_state(loss)
        auc_metric.update_state(label, score)
        precision_metric.update_state(label, score)
```

```
        recall_metric.update_state(label, score)


    def get_metric_results():
        return loss_mean_metric.result(), auc_metric.result(), precision_metric.result(),
recall_metric.result()


    @tf.function
    def train_batch(ui, label):
        with tf.GradientTape() as tape:
            score = model(ui, training=True)
            loss = loss_object(label, score) + sum(model.losses)
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
        update_metrics(loss, label, score)


    @tf.function
    def test_batch(ui, label):
        score = model(ui)
        loss = loss_object(label, score) + sum(model.losses)
        update_metrics(loss, label, score)


    for epoch in range(epochs):
        epoch_start_time = time.time()


        reset_metrics()
        for ui, label in train_ds:
            train_batch(ui, label)
        train_loss, train_auc, train_precision, train_recall = get_metric_results()


        reset_metrics()
        for ui, label in test_ds:
            test_batch(ui, label)
        test_loss, test_auc, test_precision, test_recall = get_metric_results()


        log(epoch, train_loss, train_auc, train_precision, train_recall, test_loss, test_auc, test_precision,
test_recall)
        if topk_data:
            topk(topk_data, score_fn)
        print('epoch_time=', time.time() - epoch_start_time, 's', sep='')
```

algorithm/KGCN/tool.py

```
from Recommender_System.utility.decorator import logger
from typing import List, Tuple, Dict
from collections import defaultdict
import numpy as np


@logger('根据知识图谱结构构建无向图')
def construct_undirected_kg(kg: List[Tuple[int, int, int]]) -> Dict[int, List[Tuple[int, int]]]:
    kg_dict = defaultdict(list)
    for head_id, relation_id, tail_id in kg:
        kg_dict[head_id].append((relation_id, tail_id))
        kg_dict[tail_id].append((relation_id, head_id))  # 将知识图谱视为无向图
    return kg_dict


@logger('根据知识图谱无向图构建邻接表 · ', ('n_entity', 'neighbor_size'))
def get_adj_list(kg_dict: Dict[int, List[Tuple[int, int]]], n_entity: int, neighbor_size: int) ->\
        Tuple[List[List[int]], List[List[int]]]:
    adj_entity, adj_relation = [None for _ in range(n_entity)], [None for _ in range(n_entity)]
    for entity_id in range(n_entity):
        neighbors = kg_dict[entity_id]
        n_neighbor = len(neighbors)
        sample_indices = np.random.choice(range(n_neighbor), size=neighbor_size, replace=n_neighbor <
neighbor_size)
        adj_relation[entity_id] = [neighbors[i][0] for i in sample_indices]
        adj_entity[entity_id] = [neighbors[i][1] for i in sample_indices]
    return adj_entity, adj_relation
```

algorithm/MKR/layer.py

```
import tensorflow as tf


class CrossLayer(tf.keras.layers.Layer):
    def call(self, inputs):
        v, e = inputs  # (batch, dim)
        v = tf.expand_dims(v, axis=2)  # (batch, dim, 1)
```

```
        e = tf.expand_dims(e, axis=1)  # (batch, 1, dim)
        c_matrix = tf.matmul(v, e)  # (batch, dim, dim)
        c_matrix_t = tf.transpose(c_matrix, perm=[0, 2, 1])  # (batch, dim, dim)
        return c_matrix, c_matrix_t


class CompressLayer(tf.keras.layers.Layer):
    def __init__(self, weight_regularizer, **kwargs):
        super(CompressLayer, self).__init__(**kwargs)
        self.weight_regularizer = tf.keras.regularizers.get(weight_regularizer)

    def build(self, input_shape):
        self.dim = input_shape[0][-1]
        self.weight = self.add_weight(shape=(self.dim, 1), regularizer=self.weight_regularizer,
name='weight')
        self.weight_t = self.add_weight(shape=(self.dim, 1), regularizer=self.weight_regularizer,
name='weight_t')
        self.bias = self.add_weight(shape=self.dim, initializer='zeros', name='bias')

    def call(self, inputs):
        c_matrix, c_matrix_t = inputs  # (batch, dim, dim)

        c_matrix = tf.reshape(c_matrix, shape=[-1, self.dim])  # (batch * dim, dim)
        c_matrix_t = tf.reshape(c_matrix_t, shape=[-1, self.dim])  # (batch * dim, dim)

        return tf.reshape(tf.matmul(c_matrix, self.weight) + tf.matmul(c_matrix_t, self.weight_t),
                          shape=[-1, self.dim]) + self.bias  # (batch, dim)


def cross_compress_unit(inputs, weight_regularizer):
    cross_feature_matrix = CrossLayer()(inputs)

    v_out = CompressLayer(weight_regularizer)(cross_feature_matrix)
    e_out = CompressLayer(weight_regularizer)(cross_feature_matrix)

    return v_out, e_out
```

algorithm/MKR/main.py

```
if __name__ == '__main__':
```

```
    import Recommender_System.utility.gpu_memory_growth
    from tensorflow.keras.optimizers import Adam
    from Recommender_System.data import kg_loader, data_process
    from Recommender_System.algorithm.MKR.model import MKR_model
    from Recommender_System.algorithm.MKR.train import train


    n_user, n_item, n_entity, n_relation, train_data, test_data, kg, topk_data =
data_process.pack_kg(kg_loader.ml1m_kg20k, keep_all_head=False, negative_sample_threshold=4)
    model_rs, model_kge = MKR_model(n_user, n_item, n_entity, n_relation, dim=8, L=1, H=1, l2=1e-6)
    train(model_rs, model_kge, train_data, test_data, kg, topk_data, kge_interval=3,
          optimizer_rs=Adam(0.02), optimizer_kge=Adam(0.01), epochs=20, batch=4096)


    '''
    n_user, n_item, n_entity, n_relation, train_data, test_data, kg, topk_data =
data_process.pack_kg(kg_loader.lastfm_kg15k, keep_all_head=False)
    model_rs, model_kge = MKR_model(n_user, n_item, n_entity, n_relation, dim=4, L=2, H=1, l2=1e-6)
    train(model_rs, model_kge, train_data, test_data, kg, topk_data, kge_interval=2,
          optimizer_rs=Adam(1e-3), optimizer_kge=Adam(2e-4), epochs=10, batch=256)
    '''
    '''
    n_user, n_item, n_entity, n_relation, train_data, test_data, kg, topk_data =
data_process.pack_kg(kg_loader.bx_kg20k, keep_all_head=False)
    model_rs, model_kge = MKR_model(n_user, n_item, n_entity, n_relation, dim=8, L=1, H=1, l2=1e-6)
    train(model_rs, model_kge, train_data, test_data, kg, topk_data, kge_interval=2,
          optimizer_rs=Adam(2e-4), optimizer_kge=Adam(2e-5), epochs=10, batch=32)
    '''
```

Algorithm/MKR/model.py

```
from typing import Tuple
import tensorflow as tf
from Recommender_System.algorithm.MKR.layer import cross_compress_unit
from Recommender_System.utility.decorator import logger


@logger('初始化MKR模型：', ('n_user', 'n_item', 'n_entity', 'n_relation', 'dim', 'L', 'H', 'l2'))
def MKR_model(n_user: int, n_item: int, n_entity: int, n_relation: int, dim=8, L=1, H=1, l2=1e-6) ->
Tuple[tf.keras.Model, tf.keras.Model]:
    l2 = tf.keras.regularizers.l2(l2)


    user_id = tf.keras.Input(shape=(), name='user_id', dtype=tf.int32)
```

```
    item_id = tf.keras.Input(shape=(), name='item_id', dtype=tf.int32)
    head_id = tf.keras.Input(shape=(), name='head_id', dtype=tf.int32)
    relation_id = tf.keras.Input(shape=(), name='relation_id', dtype=tf.int32)
    tail_id = tf.keras.Input(shape=(), name='tail_id', dtype=tf.int32)

    user_embedding = tf.keras.layers.Embedding(n_user, dim, embeddings_regularizer=l2)
    item_embedding = tf.keras.layers.Embedding(n_item, dim, embeddings_regularizer=l2)
    entity_embedding = tf.keras.layers.Embedding(n_entity, dim, embeddings_regularizer=l2)
    relation_embedding = tf.keras.layers.Embedding(n_relation, dim, embeddings_regularizer=l2)

    u = user_embedding(user_id)
    i = item_embedding(item_id)
    h = entity_embedding(head_id)
    r = relation_embedding(relation_id)
    t = entity_embedding(tail_id)

    for _ in range(L):
        u = tf.keras.layers.Dense(dim, activation='relu', kernel_regularizer=l2)(u)
        i, h = cross_compress_unit(inputs=(i, h), weight_regularizer=l2)
        t = tf.keras.layers.Dense(dim, activation='relu', kernel_regularizer=l2)(t)

    #rs = tf.concat([u, i], axis=1)
    rs = tf.keras.activations.sigmoid(tf.reduce_sum(u * i, axis=1, keepdims=True))
    kge = tf.concat([h, r], axis=1)
    for _ in range(H - 1):
        #rs = tf.keras.layers.Dense(dim * 2, activation='relu', kernel_regularizer=reg_l2(l2))(rs)
        kge = tf.keras.layers.Dense(dim * 2, activation='relu', kernel_regularizer=l2)(kge)
    #rs = tf.keras.layers.Dense(1, activation='sigmoid', kernel_regularizer=reg_l2(l2))(rs)
    kge = tf.keras.layers.Dense(dim, activation='sigmoid', kernel_regularizer=l2)(kge)
    kge = -tf.keras.activations.sigmoid(tf.reduce_sum(t * kge, axis=1))
    return tf.keras.Model(inputs=[user_id, item_id, head_id], outputs=rs),\
           tf.keras.Model(inputs=[item_id, head_id, relation_id, tail_id], outputs=kge)


if __name__ == '__main__':
    rs_model, kge_model = MKR_model(2, 2, 2, 2)
    u = tf.constant([0, 1])
    i = tf.constant([1, 0])
```

```
    h = tf.constant([0, 1])
    r = tf.constant([1, 0])
    t = tf.constant([0, 1])
    print(rs_model({'user_id': u, 'item_id': i, 'head_id': h}))
    print(kge_model({'item_id': i, 'head_id': h, 'relation_id': r, 'tail_id': t}))

    ds = tf.data.Dataset.from_tensor_slices(({'item_id': i, 'head_id': h, 'relation_id': r, 'tail_id': t},
tf.constant([0] * 2))).batch(2)
    kge_model.compile(optimizer='adam', loss=lambda y_true, y_pre: y_pre)
    kge_model.fit(ds, epochs=3)

    #ds = tf.data.Dataset.from_tensor_slices(({'user_id': u, 'item_id': i, 'head_id': h}, tf.constant([0.,
1.]))).batch(2)
    #rs_model.compile(optimizer='adam', loss=tf.keras.losses.BinaryCrossentropy())
    #rs_model.fit(ds, epochs=3)
```

algorithm/MKR/train.py

```
from typing import List, Tuple
import tensorflow as tf
from Recommender_System.algorithm.train import RsCallback
from Recommender_System.utility.evaluation import TopkData
from Recommender_System.utility.decorator import logger


class _KgeCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        tf.print('KGE: epoch=', epoch + 1, ', loss=', logs['loss'], sep='')


def _get_score_fn(model):
    @tf.function(experimental_relax_shapes=True)
    def _fast_model(inputs):
        return tf.squeeze(model(inputs))

    def _score_fn(inputs):
        inputs = {k: tf.constant(v, dtype=tf.int32) for k, v in inputs.items()}
        inputs['head_id'] = inputs['item_id']
        return _fast_model(inputs).numpy()
    return _score_fn
```

```
@logger('开始训练，', ('epochs', 'batch'))
def train(model_rs: tf.keras.Model, model_kge: tf.keras.Model, train_data: List[Tuple[int, int, int]],
          test_data: List[Tuple[int, int, int]], kg: List[Tuple[int, int, int]], topk_data: TopkData,
          optimizer_rs=None, optimizer_kge=None, kge_interval=3, epochs=100, batch=512):
    if optimizer_rs is None:
        optimizer_rs = tf.keras.optimizers.Adam()
    if optimizer_kge is None:
        optimizer_kge = tf.keras.optimizers.Adam()

    def xy(data):
        user_id = tf.constant([d[0] for d in data], dtype=tf.int32)
        item_id = tf.constant([d[1] for d in data], dtype=tf.int32)
        head_id = tf.constant([d[1] for d in data], dtype=tf.int32)
        label = tf.constant([d[2] for d in data], dtype=tf.float32)
        return {'user_id': user_id, 'item_id': item_id, 'head_id': head_id}, label

    def xy_kg(kg):
        item_id = tf.constant([d[0] for d in kg], dtype=tf.int32)
        head_id = tf.constant([d[0] for d in kg], dtype=tf.int32)
        relation_id = tf.constant([d[1] for d in kg], dtype=tf.int32)
        tail_id = tf.constant([d[2] for d in kg], dtype=tf.int32)
        label = tf.constant([0] * len(kg), dtype=tf.float32)
        return {'item_id': item_id, 'head_id': head_id, 'relation_id': relation_id, 'tail_id': tail_id}, label

    train_ds = tf.data.Dataset.from_tensor_slices(xy(train_data)).shuffle(len(train_data)).batch(batch)
    test_ds = tf.data.Dataset.from_tensor_slices(xy(test_data)).batch(batch)
    kg_ds = tf.data.Dataset.from_tensor_slices(xy_kg(kg)).shuffle(len(kg)).batch(batch)

    model_rs.compile(optimizer=optimizer_rs, loss='binary_crossentropy', metrics=['AUC', 'Precision',
'Recall'])
    model_kge.compile(optimizer=optimizer_kge, loss=lambda y_true, y_pre: y_pre)

    for epoch in range(epochs):
        model_rs.fit(train_ds, epochs=epoch + 1, verbose=0, validation_data=test_ds,
                     callbacks=[RsCallback(topk_data, _get_score_fn(model_rs))], initial_epoch=epoch)
        if epoch % kge_interval == 0:
```

```
        model_kge.fit(kg_ds, epochs=epoch + 1, verbose=0, callbacks=[_KgeCallback()],
initial_epoch=epoch)
```

algorithm/common.py

```
from typing import List, Callable, Dict

from Recommender_System.utility.evaluation import TopkData, topk_evaluate



def log(epoch, train_loss, train_auc, train_precision, train_recall, test_loss, test_auc, test_precision,
test_recall):
    train_f1 = 2. * train_precision * train_recall / pr if (pr := train_precision + train_recall) else 0
    test_f1 = 2. * test_precision * test_recall / pr if (pr := test_precision + test_recall) else 0
    print('epoch=%d, train_loss=%.5f, train_auc=%.5f, train_f1=%.5f, test_loss=%.5f, test_auc=%.5f,
test_f1=%.5f' %
          (epoch + 1, train_loss, train_auc, train_f1, test_loss, test_auc, test_f1))



def topk(topk_data: TopkData, score_fn: Callable[[Dict[str, List[int]]], List[float]], ks=[10, 36, 100]):
    precisions, recalls = topk_evaluate(topk_data, score_fn, ks)
    for k, precision, recall in zip(ks, precisions, recalls):
        f1 = 2. * precision * recall / pr if (pr := precision + recall) else 0
        print('[k=%d, precision=%.3f%%, recall=%.3f%%, f1=%.3f%%]' %
              (k, 100. * precision, 100. * recall, 100. * f1), end='')
    print()
```

algorithm/train.py (not-a-script meaning a dependency file)

```
from typing import List, Tuple, Callable, Dict
import tensorflow as tf
from Recommender_System.algorithm.common import log, topk
from Recommender_System.utility.evaluation import TopkData
from Recommender_System.utility.decorator import logger



def prepare_ds(train_data: List[Tuple[int, int, int]], test_data: List[Tuple[int, int, int]],
               batch: int) -> Tuple[tf.data.Dataset, tf.data.Dataset]:
    def xy(data):
        user_ids = tf.constant([d[0] for d in data], dtype=tf.int32)
        item_ids = tf.constant([d[1] for d in data], dtype=tf.int32)
        labels = tf.constant([d[2] for d in data], dtype=tf.keras.backend.floatx())
```

```
        return {'user_id': user_ids, 'item_id': item_ids}, labels

    train_ds = tf.data.Dataset.from_tensor_slices(xy(train_data)).shuffle(len(train_data)).batch(batch)
    test_ds = tf.data.Dataset.from_tensor_slices(xy(test_data)).batch(batch)

    return train_ds, test_ds



def _evaluate(model, dataset, loss_object, mean_metric=tf.keras.metrics.Mean(),
auc_metric=tf.keras.metrics.AUC(),
              precision_metric=tf.keras.metrics.Precision(), recall_metric=tf.keras.metrics.Recall()):
    for metric in [mean_metric, auc_metric, precision_metric, recall_metric]:
        tf.py_function(metric.reset_states, [], [])

    @tf.function
    def evaluate_batch(ui, label):
        score = tf.squeeze(model(ui))
        loss = loss_object(label, score) + sum(model.losses)
        return score, loss

    for ui, label in dataset:
        score, loss = evaluate_batch(ui, label)

        mean_metric.update_state(loss)
        auc_metric.update_state(label, score)
        precision_metric.update_state(label, score)
        recall_metric.update_state(label, score)

    return mean_metric.result(), auc_metric.result(), precision_metric.result(), recall_metric.result()



def _train_graph(model, train_ds, test_ds, topk_data, optimizer, loss_object, epochs):
    score_fn = get_score_fn(model)

    @tf.function
    def train_batch(ui, label):
        with tf.GradientTape() as tape:
            score = tf.squeeze(model(ui, training=True))
            loss = loss_object(label, score) + sum(model.losses)
```

```
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    for epoch in range(epochs):
        for ui, label in train_ds:
            train_batch(ui, label)

        train_loss, train_auc, train_precision, train_recall = _evaluate(model, train_ds, loss_object)
        test_loss, test_auc, test_precision, test_recall = _evaluate(model, test_ds, loss_object)

        log(epoch, train_loss, train_auc, train_precision, train_recall, test_loss, test_auc, test_precision,
test_recall)
        topk(topk_data, score_fn)


def _train_eager(model, train_ds, test_ds, topk_data, optimizer, loss_object, epochs):
    model.compile(optimizer=optimizer, loss=loss_object, metrics=['AUC', 'Precision', 'Recall'])
    model.fit(train_ds, epochs=epochs, verbose=0, validation_data=test_ds,
              callbacks=[RsCallback(topk_data, get_score_fn(model))])


class RsCallback(tf.keras.callbacks.Callback):
    def __init__(self, topk_data: TopkData, score_fn: Callable[[Dict[str, List[int]]], List[float]]):
        super(RsCallback, self).__init__()
        self.topk_data = topk_data
        self.score_fn = score_fn

    def on_epoch_end(self, epoch, logs=None):
        log(epoch, logs['loss'], logs['auc'], logs['precision'], logs['recall'],
            logs['val_loss'], logs['val_auc'], logs['val_precision'], logs['val_recall'])

        topk(self.topk_data, self.score_fn)


@logger('开始训练．', ('epochs', 'batch', 'execution'))
def train(model: tf.keras.Model, train_data: List[Tuple[int, int, int]], test_data: List[Tuple[int, int, int]],
          topk_data: TopkData, optimizer=None, loss_object=None, epochs=100, batch=512,
execution='eager') -> None:
    """
```

```
    通用训练流程。

    :param model: 模型
    :param train_data: 训练集
    :param test_data: 测试集
    :param topk_data: 用于topk评估数据
    :param optimizer: 优化器，默认为Adam
    :param loss_object: 损失函数，默认为BinaryCrossentropy
    :param epochs: 迭代次数
    :param batch: 批数量
    :param execution: 执行模式，为eager或graph。在eager模式下，用model.fit；在graph模式下，用tf.function和GradientTape
    """
    if optimizer is None:
        optimizer = tf.keras.optimizers.Adam()
    if loss_object is None:
        loss_object = tf.keras.losses.BinaryCrossentropy()

    train_ds, test_ds = prepare_ds(train_data, test_data, batch)
    train_fn = _train_eager if execution == 'eager' else _train_graph
    train_fn(model, train_ds, test_ds, topk_data, optimizer, loss_object, epochs)


@logger('开始测试，', ('batch',))
def test(model: tf.keras.Model, train_data: List[Tuple[int, int, int]], test_data: List[Tuple[int, int, int]],
         topk_data: TopkData, loss_object=None, batch=512) -> None:
    """
    通用测试流程。

    :param model: 模型
    :param train_data: 训练集
    :param test_data: 测试集
    :param topk_data: 用于topk评估数据
    :param loss_object: 损失函数，默认为BinaryCrossentropy
    :param batch: 批数量
```

```
    """
    if loss_object is None:
        loss_object = tf.keras.losses.BinaryCrossentropy()

    train_ds, test_ds = prepare_ds(train_data, test_data, batch)
    train_loss, train_auc, train_precision, train_recall = _evaluate(model, train_ds, loss_object)
    test_loss, test_auc, test_precision, test_recall = _evaluate(model, test_ds, loss_object)
    log(-1, train_loss, train_auc, train_precision, train_recall, test_loss, test_auc, test_precision,
test_recall)
    topk(topk_data, get_score_fn(model))


def get_score_fn(model):
    @tf.function(experimental_relax_shapes=True)
    def _fast_model(ui):
        return tf.squeeze(model(ui))

    def score_fn(ui):
        ui = {k: tf.constant(v, dtype=tf.int32) for k, v in ui.items()}
        return _fast_model(ui).numpy()

    return score_fn
```

data/data_loader.py

```
import os
from typing import List, Callable, Tuple
from Recommender_System.utility.decorator import logger

# 记下ds文件夹的路径，确保其它py文件调用时读文件路径正确
ds_path = os.path.join(os.path.dirname(__file__), 'ds')


def _read_ml(relative_path: str, separator: str) -> List[Tuple[int, int, int, int]]:
    data = []
    with open(os.path.join(ds_path, relative_path), 'r') as f:
        for line in f.readlines():
            values = line.strip().split(separator)
            user_id, movie_id, rating, timestamp = int(values[0]), int(values[1]), int(values[2]), int(values[3])
            data.append((user_id, movie_id, rating, timestamp))
    return data


def _read_ml100k() -> List[Tuple[int, int, int, int]]:
    return _read_ml('ml-100k/u.data', '\t')


def _read_ml1m() -> List[Tuple[int, int, int, int]]:
    return _read_ml('ml-1m/ratings.dat', '::')


def _read_ml20m() -> List[Tuple[int, int, float, int]]:
    data = []
    with open(os.path.join(ds_path, 'ml-20m/ratings.csv'), 'r') as f:
        for line in f.readlines()[1:]:
            values = line.strip().split(',')
            user_id, movie_id, rating, timestamp = int(values[0]), int(values[1]), float(values[2]), int(values[3])
            data.append((user_id, movie_id, rating, timestamp))
    return data


def _read_lastfm() -> List[Tuple[int, int, int]]:
```

```
    data = []
    with open(os.path.join(ds_path, 'lastfm-2k/user_artists.dat'), 'r') as f:
        for line in f.readlines()[1:]:
            values = line.strip().split('\t')
            user_id, artist_id, weight = int(values[0]), int(values[1]), int(values[2])
            data.append((user_id, artist_id, weight))
    return data


def _read_book_crossing() -> List[Tuple[int, str, int]]:
    data = []
    with open(os.path.join(ds_path, 'Book-Crossing/BX-Book-Ratings.csv'), 'r', encoding='utf-8') as f:
        for line in f.readlines()[1:]:
            values = line.strip().split(';')
            user_id, book_id, rating = int(values[0][1:-1]), values[1][1:-1], int(values[2][1:-1])
            data.append((user_id, book_id, rating))
    return data


@logger('开始读数据，', ('data_name', 'expect_length', 'expect_user', 'expect_item'))
def _load_data(read_data_fn: Callable[[], List[tuple]], expect_length: int, expect_user: int, expect_item: int,
               data_name: str) -> List[tuple]:
    data = read_data_fn()
    n_user, n_item = len(set(d[0] for d in data)), len(set(d[1] for d in data))
    assert len(data) == expect_length, data_name + ' length ' + str(len(data)) + ' != ' + str(expect_length)
    assert n_user == expect_user, data_name + ' user ' + str(n_user) + ' != ' + str(expect_user)
    assert n_item == expect_item, data_name + ' item ' + str(n_item) + ' != ' + str(expect_item)
    return data


def ml100k() -> List[Tuple[int, int, int, int]]:
    return _load_data(_read_ml100k, 100000, 943, 1682, 'ml100k')


def ml1m() -> List[Tuple[int, int, int, int]]:
    return _load_data(_read_ml1m, 1000209, 6040, 3706, 'ml1m')
```

```
def ml20m() -> List[Tuple[int, int, float, int]]:
    return _load_data(_read_ml20m, 20000263, 138493, 26744, 'ml20m')


def lastfm() -> List[Tuple[int, int, int]]:
    return _load_data(_read_lastfm, 92834, 1892, 17632, 'lastfm')


def book_crossing() -> List[Tuple[int, str, int]]:
    return _load_data(_read_book_crossing, 1149780, 105283, 340555, 'Book-Crossing')


# 测试数据读的是否正确
if __name__ == '__main__':
    data = book_crossing()
```

data/data_process.py

```
import os
import random
import numpy as np
from typing import Tuple, List, Callable
from collections import defaultdict
from Recommender_System.utility.evaluation import TopkData
from Recommender_System.utility.decorator import logger


@logger('开始采集负样本，', ('ratio', 'threshold', 'method'))
def negative_sample(data: List[tuple], ratio=1, threshold=0, method='random') -> List[tuple]:
    """
    采集负样本
    保证了每个用户都有正样本，但是不保证每个物品都有正样本，可能会减少用户数量和物品数量

    :param data: 原数据，至少有三列，第一列是用户id，第二列是物品id，第三列是权重
    :param ratio: 负正样本比例
```

```
    :param threshold: 权重阈值，权重大于或者等于此值为正样例，小于此值既不是正样例也不是负样例
    :param method: 采集方式，random是均匀随机采集，popular是按流行度随机采集
    :return: 带上负样本的数据集
    """
    # 负样本采集权重
    if method == 'random':
        negative_sample_weight = {d[1]: 1 for d in data}
    elif method == 'popular':
        negative_sample_weight = {d[1]: 0 for d in data}
        for d in data:
            negative_sample_weight[d[1]] += 1
    else:
        raise ValueError("参数method必须是'random'或'popular'")

    # 得到每个用户正样本与非正样本集合
    user_positive_set, user_unpositive_set = defaultdict(set), defaultdict(set)
    for d in data:
        user_id, item_id, weight = d[0], d[1], d[2]
        (user_positive_set if weight >= threshold else user_unpositive_set)[user_id].add(item_id)

    # 仅为有正样例的用户采集负样例
    user_list = list(user_positive_set.keys())
    arg_positive_set = [user_positive_set[user_id] for user_id in user_list]
    arg_unpositive_set = [user_unpositive_set[user_id] for user_id in user_list]
    from concurrent.futures import ProcessPoolExecutor
    with ProcessPoolExecutor(max_workers=os.cpu_count()//2, initializer=_negative_sample_init, initargs=(ratio, negative_sample_weight)) as executor:
        sampled_negative_items = executor.map(_negative_sample, arg_positive_set, arg_unpositive_set, chunksize=100)

    # 构建新的数据集
    new_data = []
    for user_id, negative_items in zip(user_list, sampled_negative_items):
        new_data.extend([(user_id, item_id, 0) for item_id in negative_items])
    for user_id, positive_items in user_positive_set.items():
        new_data.extend([(user_id, item_id, 1) for item_id in positive_items])
```

```
    return new_data


def _negative_sample_init(_ratio, _negative_sample_weight):  # 用于子进程初始化全局变量
    global item_set, ratio, negative_sample_weight
    item_set, ratio, negative_sample_weight = set(_negative_sample_weight.keys()),  _ratio,
_negative_sample_weight


def _negative_sample(positive_set, unpositive_set):  # 对单个用户进行负采样
    valid_negative_list = list(item_set - positive_set - unpositive_set)  # 可以取负样例的物品id列表
    n_negative_sample = min(int(len(positive_set) * ratio), len(valid_negative_list))  # 采集负样例数量
    if n_negative_sample <= 0:
        return []

    weights = np.array([negative_sample_weight[item_id] for item_id in valid_negative_list],
dtype=np.float)
    weights /= weights.sum()  # 负样本采集权重

    # 采集n_negative_sample个负样例（通过下标采样是为了防止物品id类型从int或str变成np.int或
np.str）
    sample_indices = np.random.choice(range(len(valid_negative_list)), n_negative_sample, False,
weights)
    return [valid_negative_list[i] for i in sample_indices]


@logger('开始进行id规整化')
def neaten_id(data: List[tuple]) -> Tuple[List[Tuple[int, int, int]], int, int, dict, dict]:
    """
    对数据的用户id和物品id进行规整化，使其id变为从0开始到数量减1

    :param data: 原数据，有三列，第一列是用户id，第二列是物品id，第三列是标签
    :return: 新数据，用户数量，物品数量，用户id旧到新映射，物品id旧到新映射
    """
    new_data = []
    n_user, n_item = 0, 0
    user_id_old2new, item_id_old2new = {}, {}
```

```
    for user_id_old, item_id_old, label in data:
        if user_id_old not in user_id_old2new:
            user_id_old2new[user_id_old] = n_user
            n_user += 1
        if item_id_old not in item_id_old2new:
            item_id_old2new[item_id_old] = n_item
            n_item += 1
        new_data.append((user_id_old2new[user_id_old], item_id_old2new[item_id_old], label))
    return new_data, n_user, n_item, user_id_old2new, item_id_old2new


@logger('开始数据切分，', ('test_ratio', 'shuffle', 'ensure_positive'))
def split(data: List[tuple], test_ratio=0.4, shuffle=True, ensure_positive=False) -> Tuple[List[tuple],
List[tuple]]:
    """
    将数据切分为训练集数据和测试集数据

    :param data: 原数据，第一列为用户id，第二列为物品id，第三列为标签
    :param test_ratio: 测试集数据占比，这个值在0和1之间
    :param shuffle: 是否对原数据随机排序
    :param ensure_positive: 是否确保训练集每个用户都有正样例
    :return: 训练集数据和测试集数据
    """
    if shuffle:
        random.shuffle(data)
    n_test = int(len(data) * test_ratio)
    test_data, train_data = data[:n_test], data[n_test:]

    if ensure_positive:
        user_set = {d[0] for d in data} - {user_id for user_id, _, label in train_data if label == 1}
        if len(user_set) > 0:
            print('警告：为了确保训练集数据每个用户都有正样例，%d(%f%%)条数据从测试集随机插入
训练集'
                  % (len(user_set), 100 * len(user_set) / len(data)))

        i = len(test_data) - 1
        while len(user_set) > 0:
```

```
            assert i >= 0, '无法确保训练集每个用户都有正样例，因为存在没有正样例的用户：' +
str(user_set)
            if test_data[i][0] in user_set and test_data[i][2] == 1:
                user_set.remove(test_data[i][0])
                train_data.insert(random.randint(0, len(train_data)), test_data.pop(i))
            i -= 1

    return train_data, test_data


@logger('开始准备topk评估数据，', ('n_sample_user',))
def prepare_topk(train_data: List[Tuple[int, int, int]], test_data: List[Tuple[int, int, int]],
                 n_user: int, n_item: int, n_sample_user=None) -> TopkData:
    """
    准备用于topk评估的数据

    :param train_data: 训练集数据，有三列，分别是user_id, item_id, label
    :param test_data: 测试集数据，有三列，分别是user_id, item_id, label
    :param n_user: 用户数量
    :param n_item: 物品数量
    :param n_sample_user: 用户取样数量，为None则表示采样所有用户
    :return: 用于topk评估的数据，类型为TopkData，其包括在测试集里每个用户的（可推荐物品集
合）与（有行为物品集合）
    """
    if n_sample_user is None or n_sample_user > n_user:
        n_sample_user = n_user

    user_set = np.random.choice(range(n_user), n_sample_user, False)

    def get_user_item_set(data: List[Tuple[int, int, int]], only_positive=False):
        user_item_set = {user_id: set() for user_id in user_set}
        for user_id, item_id, label in data:
            if user_id in user_set and (not only_positive or label == 1):
                user_item_set[user_id].add(item_id)
        return user_item_set

    test_user_item_set = {user_id: set(range(n_item)) - item_set
```

```
                for user_id, item_set in get_user_item_set(train_data).items()}
    test_user_positive_item_set = get_user_item_set(test_data, only_positive=True)
    return TopkData(test_user_item_set, test_user_positive_item_set)


def pack(data_loader_fn: Callable[[], List[tuple]],
         negative_sample_ratio=1, negative_sample_threshold=0, negative_sample_method='random',
         split_test_ratio=0.4, shuffle_before_split=True, split_ensure_positive=False,
         topk_sample_user=300) -> Tuple[int, int, List[Tuple[int, int, int]], List[Tuple[int, int, int]], TopkData]:
    """
    读数据，负采样，训练集测试集切分，准备TopK评估数据

    :param data_loader_fn: data_loader里面的读数据函数
    :param negative_sample_ratio: 负正样本比例，为0代表不采样
    :param negative_sample_threshold: 负采样的权重阈值，权重大于或者等于此值为正样例，小于此值既不是正样例也不是负样例
    :param negative_sample_method: 负采样方法，值为'random'或'popular'
    :param split_test_ratio: 切分时测试集占比，这个值在0和1之间
    :param shuffle_before_split: 切分前是否对数据集随机顺序
    :param split_ensure_positive: 切分时是否确保训练集每个用户都有正样例
    :param topk_sample_user: 用来计算TopK指标时用户采样数量，为None则表示采样所有用户
    :return: 用户数量，物品数量，训练集，测试集，用于TopK评估数据
    """
    data = data_loader_fn()
    if negative_sample_ratio > 0:
        data = negative_sample(data, negative_sample_ratio, negative_sample_threshold, negative_sample_method)
    else:
        data = [(d[0], d[1], 1) for d in data]  # 变成隐反馈数据
    data, n_user, n_item, _, _ = neaten_id(data)
    train_data, test_data = split(data, split_test_ratio, shuffle_before_split, split_ensure_positive)
    topk_data = prepare_topk(train_data, test_data, n_user, n_item, topk_sample_user)
    return n_user, n_item, train_data, test_data, topk_data


def pack_kg(kg_loader_config: Tuple[str, Callable[[], List[tuple]], type], keep_all_head=True,
```

```
        negative_sample_ratio=1, negative_sample_threshold=0, negative_sample_method='random',
        split_test_ratio=0.4, shuffle_before_split=True, split_ensure_positive=False,
        topk_sample_user=100) -> Tuple[int, int, int, int, List[Tuple[int, int, int]],
                                       List[Tuple[int, int, int]], List[Tuple[int, int, int]], TopkData]:
    """
    联合读数据和知识图谱，训练集测试集切分，准备TopK评估数据

    :param kg_loader_config: kg_loader里面的读知识图谱配置
    :param keep_all_head: 若为False，则读取知识图谱结构时，删除头实体在数据集里面没有对应物品的三元组
    :param negative_sample_ratio: 负正样本比例，为0代表不采样
    :param negative_sample_threshold: 负采样的权重阈值，权重大于或者等于此值为正样例，小于此值既不是正样例也不是负样例
    :param negative_sample_method: 负采样方法，值为'random'或'popular'
    :param split_test_ratio: 切分时测试集占比，这个值在0和1之间
    :param shuffle_before_split: 切分前是否对数据集随机顺序
    :param split_ensure_positive: 切分时是否确保训练集每个用户都有正样例
    :param topk_sample_user: 用来计算TopK指标时用户采样数量，为None则表示采样所有用户
    :return: 用户数量，物品数量，实体数量，关系数量，训练集，测试集，知识图谱，用于TopK评估数据
    """
    from Recommender_System.data.kg_loader import _read_data_with_kg
    data, kg, n_user, n_item, n_entity, n_relation = _read_data_with_kg(
        kg_loader_config, negative_sample_ratio, negative_sample_threshold, negative_sample_method, keep_all_head)
    train_data, test_data = split(data, split_test_ratio, shuffle_before_split, split_ensure_positive)
    topk_data = prepare_topk(train_data, test_data, n_user, n_item, topk_sample_user)
    return n_user, n_item, n_entity, n_relation, train_data, test_data, kg, topk_data
```

data/kg_loader.py

```
import os
from typing import Dict, List, Tuple, Callable, Any
from Recommender_System.data import data_loader, data_process
from Recommender_System.utility.decorator import logger

# 记下kg文件夹的路径，确保其它py文件调用时读文件路径正确
```

```
kg_path = os.path.join(os.path.dirname(__file__), 'kg')



@logger('开始读物品实体映射关系．', ('kg_directory', 'item_id_type'))
def _read_item_id2entity_id_file(kg_directory: str, item_id_type: type = int) -> Tuple[Dict[Any, int],
Dict[int, Any]]:
    item_to_entity = {}
    entity_to_item = {}
    with open(os.path.join(kg_path, kg_directory, 'item_id2entity_id.txt')) as f:
        for line in f.readlines():
            values = line.strip().split('\t')
            item_id = values[0] if item_id_type == str else item_id_type(values[0])
            entity_id = int(values[1])
            item_to_entity[item_id] = entity_id
            entity_to_item[entity_id] = item_id
    return item_to_entity, entity_to_item



@logger('开始读知识图谱结构图．', ('kg_directory', 'keep_all_head',))
def _read_kg_file(kg_directory: str, entity_id_old2new: Dict[int, int], keep_all_head=True) ->\
        Tuple[List[Tuple[int, int, int]], int, int]:
    n_entity = len(entity_id_old2new)
    relation_id_old2new = {}
    n_relation = 0
    kg = []
    with open(os.path.join(kg_path, kg_directory, 'kg.txt')) as f:
        for line in f.readlines():
            values = line.strip().split('\t')
            head_old, relation_old, tail_old = int(values[0]), values[1], int(values[2])


            if head_old not in entity_id_old2new:
                if keep_all_head:
                    entity_id_old2new[head_old] = n_entity
                    n_entity += 1
                else:
                    continue
            head = entity_id_old2new[head_old]
```

```
        if tail_old not in entity_id_old2new:
            entity_id_old2new[tail_old] = n_entity
            n_entity += 1
        tail = entity_id_old2new[tail_old]

        if relation_old not in relation_id_old2new:
            relation_id_old2new[relation_old] = n_relation
            n_relation += 1
        relation = relation_id_old2new[relation_old]

        kg.append((head, relation, tail))

    return kg, n_entity, n_relation


@logger('----------开始载入带知识图谱的数据集：', end_message='----------带知识图谱的数据集载入完成', log_time=False)
def _read_data_with_kg(kg_loader_config: Tuple[str, Callable[[], List[tuple]], type],
                       negative_sample_ratio=1, negative_sample_threshold=0, negative_sample_method='random',
                       keep_all_head=True) -> Tuple[List[Tuple[int, int, int]], List[Tuple[int, int, int]],
                                                    int, int, int, int]:
    kg_directory, data_loader_fn, item_id_type = kg_loader_config
    old_item_to_old_entity, old_entity_to_old_item = _read_item_id2entity_id_file(kg_directory, item_id_type)
    data = data_loader_fn()
    data = [d for d in data if d[1] in old_item_to_old_entity]  # 去掉知识图谱中不存在的物品
    data = data_process.negative_sample(data, negative_sample_ratio, negative_sample_threshold, negative_sample_method)
    data, n_user, n_item, _, item_id_old2new = data_process.neaten_id(data)
    entity_id_old2new = {old_entity: item_id_old2new[old_item] for old_entity, old_item in old_entity_to_old_item.items()}
    kg, n_entity, n_relation = _read_kg_file(kg_directory, entity_id_old2new, keep_all_head)
    return data, kg, n_user, n_item, n_entity, n_relation


# kg_loader_configs: (kg_directory, data_loader_fn, item_id_type)
bx_kg20k = 'bx-kg20k', data_loader.book_crossing, str
bx_kg150k = 'bx-kg150k', data_loader.book_crossing, str
```

```
lastfm_kg15k = 'lastfm-kg15k', data_loader.lastfm, int

ml1m_kg20k = 'ml1m-kg20k', data_loader.ml1m, int

ml1m_kg1m = 'ml1m-kg1m', data_loader.ml1m, int

ml20m_kg500k = 'ml20m-kg500k', data_loader.ml20m, int



if __name__ == '__main__':

    data, kg, n_user, n_item, n_entity, n_relation = _read_data_with_kg(ml1m_kg1m)
```

utility/competition.py

```
if __name__ == '__main__':
    import Recommender_System.utility.gpu_memory_growth
    import tensorflow as tf
    from Recommender_System.data import data_loader, data_process
    from Recommender_System.algorithm.FM.model import FM_model
    from Recommender_System.algorithm.GMF.model import GMF_model
    from Recommender_System.algorithm.LFM.model import LFM_model
    from Recommender_System.algorithm.MLP.model import MLP_model
    from Recommender_System.algorithm.NeuMF.model import NeuMF_model
    from Recommender_System.algorithm.DeepFM.model import DeepFM_model
    from Recommender_System.algorithm.train import train

    n_user, n_item, train_data, test_data, topk_data = data_process.pack(data_loader.ml100k)

    dim = 16

    model = FM_model(n_user, n_item, dim=dim, l2=0)
    train(model, train_data, test_data, topk_data, epochs=10)

    model = GMF_model(n_user, n_item, dim=dim, l2=0)
    train(model, train_data, test_data, topk_data, epochs=10)

    model = LFM_model(n_user, n_item, dim=dim, l2=0)
    train(model, train_data, test_data, topk_data, loss_object=tf.losses.MeanSquaredError(), epochs=10)

    model = MLP_model(n_user, n_item, dim=dim * 2, layers=[dim * 2, dim, dim // 2], l2=0)
    train(model, train_data, test_data, topk_data, epochs=10)

    model, _, _ = NeuMF_model(n_user, n_item, gmf_dim=dim // 2, mlp_dim=dim * 2, layers=[dim * 2,
dim, dim // 2], l2=0)
    train(model, train_data, test_data, topk_data, epochs=10)

    model = DeepFM_model(n_user, n_item, dim // 2, layers=[dim, dim, dim], l2=0)
    train(model, train_data, test_data, topk_data, epochs=10)
```

utility/decorator.py

```
import time
import inspect
```

```
from functools import wraps
from typing import Tuple



def arg_value(arg_name, f, args, kwargs):
    if arg_name in kwargs:
        return kwargs[arg_name]

    i = f.__code__.co_varnames.index(arg_name)
    if i < len(args):
        return args[i]

    return inspect.signature(f).parameters[arg_name].default



def logger(begin_message: str = None, log_args: Tuple[str] = None, end_message: str = None, log_time: bool = True):
    def logger_decorator(f):
        @wraps(f)
        def decorated(*args, **kwargs):
            if begin_message is not None:
                print(begin_message, end='\n' if log_args is None else '')

            if log_args is not None:
                arg_logs = [arg_name + '=' + str(arg_value(arg_name, f, args, kwargs)) for arg_name in log_args]
                print(', '.join(arg_logs))

            start_time = time.time()
            result = f(*args, **kwargs)
            spent_time = time.time() - start_time

            if end_message is not None:
                print(end_message)

            if log_time:
                print('（耗时', spent_time, '秒）', sep='')

            return result
```

```
        return decorated
    return logger_decorator
```

utility/evaluation.py

```
from dataclasses import dataclass
from typing import Tuple, List, Callable, Dict


@dataclass
class TopkData:
    test_user_item_set: dict  # 在测试集上每个用户可以参与推荐的物品集合
    test_user_positive_item_set: dict  # 在测试集上每个用户有行为的物品集合


@dataclass
class TopkStatistic:
    hit: int = 0  # 命中数
    ru: int = 0  # 推荐数
    tu: int = 0  # 行为数


def topk_evaluate(topk_data: TopkData, score_fn: Callable[[Dict[str, List[int]]], List[float]],
                  ks=[1, 2, 5, 10, 20, 50, 100]) -> Tuple[List[float], List[float]]:
    kv = {k: TopkStatistic() for k in ks}
    for user_id, item_set in topk_data.test_user_item_set.items():
        ui = {'user_id': [user_id] * len(item_set), 'item_id': list(item_set)}
        item_score_list = list(zip(item_set, score_fn(ui)))
        sorted_item_list = [x[0] for x in sorted(item_score_list, key=lambda x: x[1], reverse=True)]

        positive_set = topk_data.test_user_positive_item_set[user_id]
        for k in ks:
            topk_set = set(sorted_item_list[:k])
            kv[k].hit += len(topk_set & positive_set)
            kv[k].ru += len(topk_set)
            kv[k].tu += len(positive_set)
    return [kv[k].hit / kv[k].ru for k in ks], [kv[k].hit / kv[k].tu for k in ks]  # precision, recall
```

utility/gpu_memory_growth.py

```
"""
import此文件后将gpu设置为显存增量模式
"""

from tensorflow import config

gpus = physical_devices = config.list_physical_devices('GPU')
if len(gpus) == 0:
    print('当前没有检测到gpu，设置显存增量模式无效。')
for gpu in gpus:
    try:
        config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)
```