## 1.args：定义了各个参数的默认值：

dataset：数据集：默认为movie

n\_epochs：轮数：默认为20

dim：用户和实体嵌入的维度，默认为8

L：低层数量：默认为1

H：高层数量：默认为1

batch\_size：批量大小，默认为4096

l2\_weight：L2正则化的权重：默认为0.000006

lr\_rs：推荐系统的学习率，默认为0.02

lr\_kge：知识图谱的学习率，默认为0.01

kge\_interval：KGE任务的训练间隔：默认为3

## 2.初始化数据data：load\_data(args)

返回的有：

[0]:user的数量：6036

[1]:item的数量：2347

[2]:entity的数量：6729

[3]:relation的数量：7

[4]:训练数据三元组

[5]:测试数据三元组

[6]:评估数据三元组

[7]:知识图谱

## 3.执行train（args,data,False,True）

### 3.1执行MKR模型：

model=MKR(args,n\_user=6036,n\_item=2347,n\_entity=6729,n\_relation=7)

3.1.1:首先赋值数据\_parse\_args，将n\_user=6036,n\_item=2347,n\_entity=6729,n\_relation=7各自赋值，然后定义了vars\_rs=[],vars\_kge=[],用于计算L2损失

3.1.2：\_build\_inputs：使用placeholder定义一些目录，包括：

user\_indices：用户目录

item\_indices：物品目录

labels：标签目录

head\_indices：头诗题目录

tail\_indices：尾实体目录

relation\_indices：关系目录

3.1.3：构建低层模型

（1）使用tf.get\_variable生成词嵌入向量矩阵：

user\_emb\_matrix:shape=(6036,8)

item\_emb\_matrix:shape=(2347,8)

entity\_emb\_matrix:shape=(6729,8)

relation\_emb\_matrix:shape=(7,8)

（2）tf.nn.embedding\_lookup根据嵌入的矩阵和目录查找对应的词嵌入向量：

寻找user\_embeddings(b,8)

item\_embeddings

head\_embeddings

relation\_embeddings

tail\_embeddings

（3）构建一层低层模型：

user\_embeddings和tail\_embeddings做全连接层处理Dense

item\_embeddings和head\_embeddings做交叉单元处理CrossCompressUnit

Dense:

input(b,8)乘上权重矩阵weight(8,8)+bias(1,8)=output(b,8)

CrossCompressUnit:

v=item\_embeddings(b,8)

e=head\_embeddings(b,8)

增加维度，v=(b,8,1),e=(b,1,8)

c\_matrix=(b,8,8)

c\_matrix\_transpose=把8和8进行转置

对这两个矩阵进行降维处理，变为(b\*8,8)

输出的v\_output,e\_output=(b\*8,8)\*(8,1)变成(b\*8,1)reshape之后变成(b,8)+bias(1,8)=(b,8)

（4）定义高层模型：

scores=(b,8)对每一行求和=(b,1)

然后对scores使用sigmoid激活函数得到scores\_normalized

得到head\_relation\_concat合并：(b,8\*2)

将head\_relation\_concat放入Dense进行全连接层运算得到tail\_pred

output=(b\*16)\*(16\*8)+bias(1\*8)=output(b\*8)

对tail\_pre使用sigmoid激活函数

scores\_kge=(b)

（5）计算损失：

推荐算法模型的损失：

对于给定的scores计算sigmoid交叉熵，然后计算所有值的平均值，得到base\_loss\_rs

对于user和item的embeddings计算l2损失，得到l2\_loss\_rs

对于每次计算的权重计算l2损失，加到l2\_loss\_rs上

最终的loss\_rs=base\_loss\_rs+l2\_loss\_rs\*l2\_weight

知识图谱同理

（6）定义优化器，寻找全局最优点的优化算法，引入了二次方梯度校正

根据刚才得到的损失，和学习率0.02，0.01，得到optimizer\_rs和optimizer\_kge

3.2设置top-k评估

3.2.1 将train\_data中的user item放入字典中

将test\_data中user对item评分的user item放入字典中

然后得到共同的user\_list

item\_list的大小为item的数量

3.3 训练推荐系统

（1）调用get\_feed\_dict\_for\_rs方法得到feed\_dict字典

其中包含user\_indices

item\_indices

labels

head\_indices

然后传入model.train\_rs进行run

4.开始训练模型

首先进行模型参数的初始化，然后设置轮数n\_epochs=20

4.1 训练推荐系统

每次取出4096个数据进行训练，一共进行110次循环

model.train\_rs(sess, get\_feed\_dict\_for\_rs(model, train\_data, start, start + args.batch\_size))

def get\_feed\_dict\_for\_rs(model, data, start, end):

    feed\_dict = {model.user\_indices: data[start:end, 0],# （4096\*1）

                 model.item\_indices: data[start:end, 1],

                 model.labels: data[start:end, 2],

                 model.head\_indices: data[start:end, 1]}

return feed\_dict

    def train\_rs(self, sess, feed\_dict):

        # 训练推荐算法模型

        return sess.run([self.optimizer\_rs, self.loss\_rs], feed\_dict)

Sess.run([rs优化器，rs损失]，用户目录、物品目录、标签目录、头实体目录)即可进行推荐系统训练

4.2 训练知识图谱

每三轮训练一次知识图谱，训练方法同理

model.train\_kge(sess, get\_feed\_dict\_for\_kge(model, kg, start, start + args.batch\_size))

def get\_feed\_dict\_for\_kge(model, kg, start, end):

    feed\_dict = {model.item\_indices: kg[start:end, 0],

                 model.head\_indices: kg[start:end, 0],

                 model.relation\_indices: kg[start:end, 1],

                 model.tail\_indices: kg[start:end, 2]}

return feed\_dict

    def train\_kge(self, sess, feed\_dict):

        # 训练知识图谱词嵌入模型

        return sess.run([self.optimizer\_kge, self.rmse], feed\_dict)

Sess.run([kge优化器，kge损失],物品目录、头实体目录、关系目录、尾实体目录)即可进行知识图谱训练

5.测试模型

使用eval进行测试，并输出

train\_auc, train\_acc = model.eval(sess, get\_feed\_dict\_for\_rs(model, train\_data, 0, train\_data.shape[0]))

eval\_auc, eval\_acc = model.eval(sess, get\_feed\_dict\_for\_rs(model, eval\_data, 0, eval\_data.shape[0]))

test\_auc, test\_acc = model.eval(sess, get\_feed\_dict\_for\_rs(model, test\_data, 0, test\_data.shape[0]))

def get\_feed\_dict\_for\_rs(model, data, start, end):

    feed\_dict = {model.user\_indices: data[start:end, 0],# （4096\*1）

                 model.item\_indices: data[start:end, 1],

                 model.labels: data[start:end, 2],

                 model.head\_indices: data[start:end, 1]}

return feed\_dict

    def eval(self, sess, feed\_dict):

        # 评估模型

        labels, scores = sess.run([self.labels, self.scores\_normalized], feed\_dict)

        # auc为曲线下面积，数值越高，模型越优秀

        auc = roc\_auc\_score(y\_true=labels, y\_score=scores)

        predictions = [1 if i >= 0.5 else 0 for i in scores]

        # mean求均值，equal输出相等为True，不相等为False

        acc = np.mean(np.equal(predictions, labels))

        return auc, acc

Sess.run([标签，标准化的分数]，用户目录、物品目录、标签目录、头实体目录)即可进行测试

6.进行top-k评估

6.1 定义准确率列表、召回率列表

6.2 对于每一个重复用户，test\_item\_list=物品集-训练数据中的物品集

对test\_item\_list进行get\_scores训练得到items和scores，对于每一个item和score，放入一个list中

然后对score进行从高到低排序。

对于每一个k\_list，添加进准确率列表和召回率列表，最终计算准确率、召回率、F1值