R&S | 手把手搞推荐[3]：数据集存取思路

原创 机智的叉烧 CS的陋室 2019-05-21

点击上方蓝色文字立刻订阅精彩

Dance of The Violins 2

F-777 - Viking Dance Machine

【R&S】

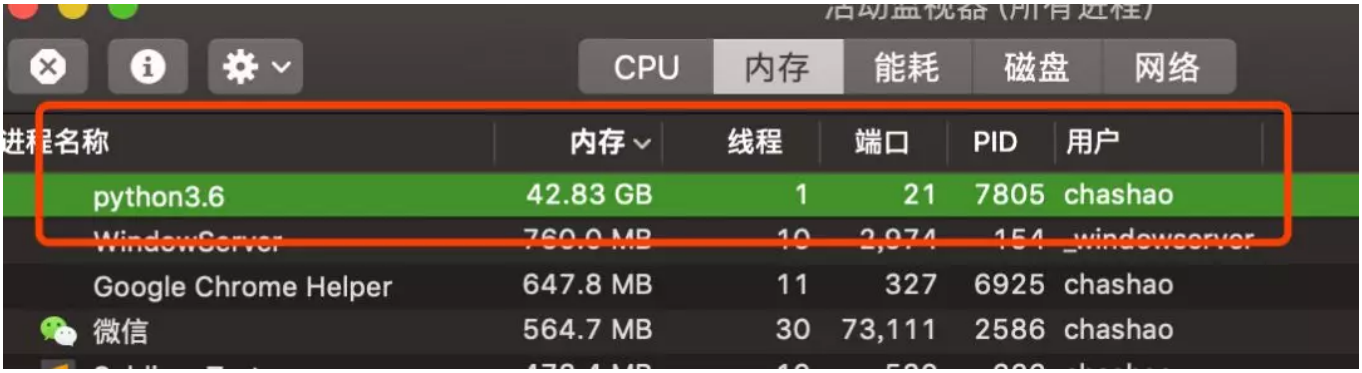
本栏目从原来的【RS】改为【R&S】，意为“recommend and search”，即推荐和搜索，结合本人近期的工作方向、最近上的七月在线的课、自己自学、而退出的特别栏目。当然，按照我往期的风格，更加倾向于去讨论一些网上其实讲得不够的东西，非常推荐大家能多看看并且讨论，欢迎大家给出宝贵意见，觉得不错请点击推文最后的好看，感谢各位的支持。

另外，【手把手搞推荐】是我近期开始的连载，结合自己所学，带上代码的手把手和大家分享一些模型和数据处理方式，欢迎关注。

往期回顾：

- R&S | 手把手搞推荐[0]：我的推荐入门小结
- R&S | 手把手搞推荐[1]：数据探索
- R&S | 手把手搞推荐[2]：特征工程指南
- RS | 推荐系统整体设计
- RS | 论文阅读：用于YouTube推荐的深度神经网络

上一期讲到我们进行一整套特征工程，然而可怕的是，事实上这样存储的训练集数据体积无敌大，这样的数据在计算图中将遇到内存不足的关键问题，给大家看看我在读取数据的时候的壮观景象：



因此，在进行LR之前，需要教大家设计更合适的方法存取数据集。

简单说思路

口述思路

现在的当务之急是找到一个合适的方式压缩存储空间，从而保证空间复杂度较低，然而在更多现实场景，其实还会涉及很多问题，因此在这里，我统一讲讲如何设计一个好的方式进行数据集的存取，说白了就是要思考这几个问题：

- 用什么存？纯文本？CSV？SQL？excel？甚至是其他更新更多样化的操作
- 用什么结构存？

那么，在进行选择的时候，实际上要考虑的是这几个问题：

- 安全性。存取安全，一定级别下防止泄露。
- 完整性。不会存在数据出错。
- 高效性。存取的复杂度要在可控范围内。

实际问题

在我们的这个问题下，安全性不要求；完整性上述方案基本能保证，所以也没问题；问题就在于高效性，现在这个是我们目前面临的重大问题，所以我们要好好处理，保证数据不失真的情况下去处理。

压缩的核心在于略去不必要的信息，或者用更简单的方式来描述更多的信息，例如条件允许的情况下二元数组可以用一元数组+函数的方式存储，那么在此处，我们先看看数据。

- 我们的数据是一套one-hot数据
- one-hot数据本身是一种稀疏矩阵的结构
- 稀疏矩阵中含有大量的0，仅有少部分非0

因此，我们可以用稀疏矩阵的方式进行存取，只记住非零位置下的值，其他位置为0即可。

稀疏矩阵存取

稀疏矩阵的有关理论在此处不赘述，可以自行查阅，此处给出一种我最终选择的方案，用CSR格式，技术方案是用scipy.sparse。

简单说说CSR，CSR格式实际上就是用一个三元组数组来表示这个稀疏矩阵，三元组分别表示(col,row,data)，即行，列，数值，非零的位置的数值得以保留，然后其他位置都是0。

存储

首先来看，用旧方案和新方案的数据结构：

> 旧数据：纯onehot，每行接近1w个数据
>
> 新数据：分为两块存储，X部分用稀疏矩阵，然后用npz存储，Y部分用one-hot存储。
>
> 0.0 0.0 1.0 0.0 0.0 23,1.0 3737,1.0 3750,1.0 3757,1.0 6249,1.0 9801,1.0 9806,1.0 9819,1.0

存储完，体积只有59M，这样读取到内存的体积也会小很多，甚至整块都读进去都没问题。

这里会用到稀疏矩阵的2个函数，此处导入：

```
from scipy.sparse import csr_matrix, save_npz
```

下面来看看怎么实现的，下面是根据之前上游合并好的数据，利用加载得到的oh_encoder，分别进行转化，分为两块输出，一方面是x特征数据矩阵，另一方面是y标签one-hot矩阵。

```
def gen_res(source_data, oh_encoder):
    col_all = []
    row_all = []
    data_all = []
    idx = 0
    y_res = []
    with open(source_data, encoding="utf8") as f:
        for line in f:
            if idx == 0:
                idx = 1
                continue
            ll = line.strip().split("::")
            ll = line.strip().split("::")
            data_item = []
            scores_item = []
            scores_item = scores_item + oh_encoder["scores"].transform([[ll[0]]])[0].tol
            data_item = data_item + oh_encoder["movie_id"].transform([[ll[1]]])[0].tolis
```

```
            data_item = data_item + oh_encoder["movie_year"].transform([[get_year(ll[2])
            data_item = data_item + get_movie_type_oh(ll[3], oh_encoder["movie_type"]).t
            data_item = data_item + oh_encoder["user_id"].transform([[ll[4]]])[0].tolist
            data_item = data_item + oh_encoder["user_gentle"].transform([[ll[5]]])[0].to
            data_item = data_item + oh_encoder["user_age"].transform([[ll[6]]])[0].tolis
            data_item = data_item + oh_encoder["user_occupation"].transform([[ll[7]]])[0
            # Y处理
            y_res.append(scores_item)
            # X处理
            col, data = sparse_list(data_item)
            col_all = col_all + col
            row_all = row_all + [idx - 1 for item in range(len(col))]
            data_all = data_all + data
            idx = idx + 1
            if idx % 10000 == 0:
                print("generating %s data items" % (idx))
                # break
    x_res = csr_matrix((data_all, (row_all, col_all)), shape=(max(row_all) + 1, 9831))
    return x_res, y_res
```

总结这么几个点：

- scores数据我还是保留着one-hot，存储差别不是很大，后面的代码也不用改太多，条件允许的话偷个懒吧
- 其他的onehot数据，我是先用transform计算出来，进行组合，然后再用sparse_list转化，此处，只需要抽取col和data，存储即可(row因为没必要存，行数后面取可以算出来)
- csr_matrix是一个转化为稀疏矩阵的函数
- 9831是除了特征维度，此处规范化，避免特征出现个数不足的现象
- 有关稀疏函数在这块的应用，非常建议大家多看看API文档，传送门在这里：https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html
- 这块的时间其实挺长的，毕竟涉及大量的处理，有条件的各位可以考虑用mapreduce分布式尝试

下面给出sparse_list的定义，这块的任务是对原来的向量，转为用[col,data]存储的形式：

```
def sparse_list(array_get):
    col = []
    data = []
    for idx in range(len(array_get)):
        if array_get[idx] != 0:
            col.append(idx)
            data.append(array_get[idx])
    return col, data
```

在gen_res得到结果后，就可以进行进一步存储，下面给出一个训练数据方面的例子。

```
# 训练数据生成
```

```
print("generating training data")
x_train, y_train = gen_res(TRAIN_DATA_PATH, oh_encoder)
with open(GEN_DATA_TRAIN_Y_PATH, "w", encoding='utf8') as f:
    for item in y_train:
        f.write("%s\n" % (",".join([str(i) for i in item])))
save_npz(GEN_DATA_TRAIN_X_PATH, x_train)
print("training data generation done")
```

大写基本是写死的路径和参数，ohencoder是使用的转化器，savenpz是存储稀疏矩阵的函数。

读取

这块本来是在后续机器学习模型那集才会说的，此处为了内容完善写出来提早放出来~下一集这个函数就不单独放出来，直接调用啦。

首先是看这里需要的重要包。

```
from scipy.sparse import load_npz
```

load_npz是对应稀疏函数的加载包。

```
def load_dataset(x_path, y_path):
    y_ = []
    y_oh = []
    idx = 0
    with open(y_path, encoding="utf8") as f:
        for line in f:
            ll = line.strip().split(",")
            y_item = max([idx * int(float(ll[idx])) for idx in range(5)]) + 1
            y_oh_item = [int(float(item)) for item in ll[:5]]
            y_.append(y_item)
            y_oh.append(y_oh_item)
            idx = idx + 1
            if idx % 10000 == 0:
                print("loading %s" % idx)
    x_ = load_npz(x_path)
    return x_, y_, y_oh
```

同样简单粗暴地给关键点吧

- 此处要用scipy给的数据结构来存
- 原因是scipy.sparse这个数据类型能够直接放在sklearn下的机器学习模型训练输入中
- 存取比较直接，不用关心内部逻辑，复杂度一般不会太高
- x和y分别取出分别放好，毕竟后续训练的时候也是要分开放入的
- 此处的输出我分了三个，作为稀疏矩阵的特征，数值型的y，和one-hot的y，日后结果评估会有用。

小结

算法从来不止是机器学习，而是对整个项目流程的把握，根据特定目标做好规划，一整个项目才得以完成。本文主要以movielens为例谈论数据集存取策略的问题，是特征工程后确定数据存储格式、存取方式的重要一步，合理高效的存取策略会令整个系统在进行模型计算时更加高效，省时省空间。

我是叉烧，欢迎关注我！

叉烧，机器学习算法实习生，北京科技大学数理学院统计学研二硕士（保研），本科北京科技大学信息与计算科学、金融工程双学位毕业，硕士期间发表论文6篇，学生一作3篇，1项国家自然科学基金面上项目学生第2参与人，参与国家级及以上学术会议4次，其中，1次优秀论文，国家奖学金。曾任去哪儿网大住宿事业部产品数据，美团点评出行事业部算法工程师。

微信 zgr950123
邮箱 chashaozgr@163.com
知乎 机智的叉烧

微信个人公众号
CS的陋室

喜欢此内容的人还喜欢

属于算法的大数据工具-pyspark：10天吃掉那只pyspark
CS的陋室

央媒聚焦国铁集团工作会议，这些关键词值得期待
中国铁路

班主任总结：2020高考成绩最惨的竟是这些学生！2021/22届考生务必警醒！