实验报告

郭丹琪 2018202067　　周珂 2018202070

一、问题分析

问题为根据SQL查询语句，预测查询规模。SQL语句的类型为范围查询与等值查询，需要根据100000条训练集预测5000条测试集的结果。

分析该问题为回归预测问题，问题主要分为两部分。第一部分是对数据的提取与特征向量的构建，需要先从数据集文件中获取需要的数据，然后再将其构建为模型可以接受的特征向量的形式。第二部分是模型的构建与训练，需要先挑选出适合该问题的模型，再在原本模型的基础上进行多次训练和测试，再根据测试结果对模型进行调整和优化。

二、数据处理

1、数据读取

定义了函数 get_data(file_path, train)，对数据进行预处理。数据集中各条数据根据 # 划分为4个部分：第一个部分是要选的表名，存的是表的别名；第二个部分是连接条件，单表查询时的数据为' '；第三个部分是where子句选择条件；第四部分是查询结果，test数据不存在第四部分。

```
data = [each for each in csv.reader(csvfile, delimiter = '#')] #划分各部分
```

数据提取部分示例：第三部分数据提取

```
        items = data[i][2].split(',')
        conditions = []
        for j in range(int(len(items) / 3)):
            condition = []
            condition.append(items[3 * j])
            condition.append(items[3 * j + 1])
            condition.append(int(items[3 * j + 2]))
            conditions.append(condition)
        data[i][2] = conditions
```

下述为读取到的数据样例：

```
#训练数据：
[['t', 'mi_idx'], #第一部分
[['t.id', 'mi_idx.movie_id']],#第二部分 单表查询该部分为空
[['t.kind_id', '=', 7], ['mi_idx.info_type_id', '>', 99]],#第三部分
283812]#第四部分

#测试数据：
[['t'], [], [['t.production_year', '>', 2004]]]
```

除了训练数据集外，还需要获取描述各表信息的数据，并对属性和表进行编号：

```
def get_column_details():
    column_details = pd.read_csv("column_min_max_vals.csv")
    table_index = {'t' : 0, 'mc' : 1, 'ci' : 2, 'mi' : 3, 'mi_idx' : 4, 'mk' :
5}
    feature_index = {}
    for i in range(len(column_details)):
        feature_index[column_details.iloc[i]['name']] = i
    return column_details, table_index, feature_index
```

2、多表连接处理

A、方法一

经过前面的预处理发现，数据只包含有5个表，20个属性，所以给每个表和属性加上了编号。采用20列的数组，将连接属性对应编号的位置为1，其它属性置为0，以此来表示SQL语句中的连接信息。

```
def create_join_array(example):
    join = []
    for i in range(len(example)):
        str1 = example[i][1]
        X_temp = np.zeros(20)
        #X_temp为每一行的连接条件的初始化数组
        for j in range(len(str1)):
            temp = str1[j]
            for k in range(len(temp)):
                temp_name = temp[k]
                num = feature_index.get(temp_name)
                X_temp[num] = 1
        join.append(X_temp)
    return join
```

例如对于提取后的数据`[['t', 'mi_idx'], [['t.id', 'mi_idx.movie_id']], [['t.kind_id', '=', 7], ['mi_idx.info_type_id', '>', 99]], 283812]`，就要将属性`'t.id', 'mi_idx.movie_id'`对应的位置上的值置为1，得到的结果示例如下。

```
[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

B、方法二

考虑到我们得到的数据大多数是两表连接，最多是三表连接，即最多涉及到4个属性，所以在上述处理方式下记录连接条件的20列向量中存在很多0，数据过于稀疏。对数据进行观察后发现，数据集中的连接关系种类很少，经过计算，数据集中只存在五种连接关系，将其对应到属性的编号，并对连接关系进行编号，得到连接关系的记录为`{(0, 15): 1, (0, 5): 2, (0, 12): 3, (0, 8): 4, (0, 18): 5}`，所以将SQL语句中的连接信息用连接关系的编号来表示，只需要两列向量（因为最多有三表连接）。

处理连接关系的过程如下：

```
def count_connection(example):
```

```
    X = []
    for i in range (len(example)):
        str1 = example[i][1]
        X_temp = np.zeros(2)
        for j in range(len(str1)):
            temp = str1[j]
            temp_name1 = temp[0]
            temp_name2 = temp[1]
            num1 = feature_index.get(temp_name1)
            num2 = feature_index.get(temp_name2)
            list1 = [num1, num2]
            key = tuple(list1)
            if key in connection:
                X_temp[j] = connection.get(key)
            else:
                value = len(connection)
                connection[key] = value + 1
                X_temp[j] = connection.get(key)
        X.append(X_temp)
    return X
```

例如在提取后的数据中`[['t', 'mi_idx'], [['t.id', 'mi_idx.movie_id']], [['t.kind_id', '=', 7], ['mi_idx.info_type_id', '>', 99]], 283812], ['t.id', 'mi_idx.movie_id']`对应的连接类型的编号为1，连接信息就表示为[1. 0.]

C、方法三

发现基于方法二处理得到的特征向量的训练效果不是很好，所以决定采用5列向量来表示连接信息，每一列的编号和连接类型编号相同，对应的位置置为1，其余位置置为0.

比如：在提取后的数据中`[['t', 'mi_idx'], [['t.id', 'mi_idx.movie_id']], [['t.kind_id', '=', 7], ['mi_idx.info_type_id', '>', 99]], 283812], ['t.id', 'mi_idx.movie_id']`对应的连接类型的编号为1，所以连接信息表示为[1. 0. 0. 0. 0.]

3、查询条件处理

共有5个表，20种属性，采用40列的向量来表示选择信息，向量格式为`[lb1, ub1, lb2, ub2, ……lbd, ubd]`，其中 lb 表示 lower bound，ub 表示 upper bound。所以对于属性i，i∈(0,19)，向量中对应表示该属性选择信息的位置为`2i，2*i + 1`。如果SQL语句中只限制了该属性的最大值，那么最小值就要在统计信息表中找到。如果为等值选择，那么最大值和最小值为同一个数。

处理对应的主要步骤如下：

```
            if temp[1] == '=': #等值：最大值和最小值为同一个数
                temp_name = temp[0]
                num = feature_index.get(temp_name)
                X_temp[num * 2] = temp[2]
                X_temp[num * 2 + 1] = temp[2]
            #非等值：没有限制的方向到column_details中获取最大或最小值
            elif temp[1] == '>':
                temp_name = temp[0]
                num = feature_index.get(temp_name)
                X_temp[num * 2] = temp[2]
```

```
                X_temp[num * 2 + 1] = column_details.iloc[num]['max']

            else :
                temp_name = temp[0]
                num = feature_index.get(temp_name)
                X_temp[num * 2 + 1] = temp[2]
                X_temp[num * 2] = column_details.iloc[num]['min']
```

对于提取后的数据`[['t', 'mi_idx'], [['t.id', 'mi_idx.movie_id']], [['t.kind_id', '=', 7], ['mi_idx.info_type_id', '>', 99]], 283812]`，对应的选择信息表示为如下：

```
[    0.     0.     7.     7.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.     0.     0.     0.     0.
     0.     0.     0.     0.     0.    99.   113.     0.     0.
     0.     0.     0.     0.]
```

4、特征归一化

对获取的特征向量进行归一化处理，防止某一维的特征过大或过小，从而在训练中起的作用不平衡。

```
from sklearn.preprocessing import StandardScaler
x = np.array(create_X_vector(train_data_x, feature_index, column_details))
scaler = StandardScaler().fit(x)
x_origin = x
x = scaler.transform(x)
```

三、模型构建

1、线性回归模型

初次尝试考虑使用简单的线性回归，利用 sklearn.linear_model 里的线性模型对训练数据进行模型构建。

LinearRegression

首先尝试基于最小二乘法的线性回归，分别对归一化后的数据和无归一化的数据都进行了测试。

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x, y)
predicts = model.predict(test_x)
```

发现无论数据有无归一化，得到的结果在测评中都表现不太好，测评分数都在5.8左右。分析原因可能是这样的线性回归模型过于简单，因此考虑尝试其他线性回归模型。

Ridge岭回归

考虑到查询涉及到的几张表和属性之间存在差异，可能对某些属性值进行选择时一点小的变动就会让查询结果数目发生很大的变化，而对有的属性值选择时的大的变动反而对查询结果数目没什么影响。了解到岭回归在最小二乘法的基础上进行改进，通过放弃最小二乘法的无偏性，在回归估计中增加额外的偏差度，有效减少方差，以损失了无偏性，来换取高的数值稳定性，从而得到较高的计算精度。因此使用岭回归进行了测试。

```
from sklearn.linear_model import Ridge
model = Ridge()
model.fit(x, y)
predicts = model.predict(test_x)
```

但是发现对有无归一化的数据的测试结果都没有明显优于简单线性回归模型，测评分数也都在5.8左右。

结果分析

之后又尝试了 sklearn.linear_model 中的部分线性回归模型，包括贝叶斯岭回归、弹性网络回归、Lasso回归模型等。发现哪怕在弹性网络回归、Lasso回归这样的惩罚回归模型中实现了对贡献较小的变量系数的相应调整，并且根据数据特点对模型进行了相应选择和调整，得到的测试分数均在5.8左右，结果都没有明确变化，而在在有无归一化的数据下测试结果都是相近的。

分析这是损失函数的问题。在线性回归模型中，损失函数都是以均方误差（MSE）为基础进行调整。对于普通的使用最小二乘法的系数估计，其依赖于模型各项的相互独立性。

$$MSE = \frac{\sum (y_{true} - y_{pred})^2}{n}$$

而测评分数使用的是均方对数损失（MSLE），是 MSE 的一个改进版本。

$$MSLE = \frac{\sum (log(y_{true} + 1) - log(y_{pred} + 1))^2}{n}$$

从形式上看，MSLE 相当于是把 MSE 中每一项的损失都缩小了。对于该预测问题，在模型中使用 MSE 作为损失函数，最终再用 MSLE 进行测评，效果不会好。因为查询结果的取值范围较大，预测结果与正确结果的差值范围就较大，哪怕预测只出现少数错误，错误差值较大的话，就会对 MSE 的值产生很大的影响，而 MSLE 中用到取对数来降低这样的影响，导致 MSE 和 MSLE 在该问题上的评判标准不同，所以用 MSE 作为损失函数训练的结果再用 MSLE 来评分的话效果自然不好。

所以这些线性回归模型的测试结果不好，一方面是因为这些模型基本都使用 MSE 作为损失函数，另一方面可能是因为该问题本身的特征值和结果之间的关系就不是线性关系，强行将其看作线性关系来处理的话结果就不会很好，而且几个线性模型得到的结果都是相近的。

在测试中还发现，数据有无归一化的结果差别不大。分析后发现这是因为该预测问题中大部分特征在维度或者单位上差别不大，基本是在同一个尺度下的，在特征空间中对样本的距离产生的影响是同级的，所以数据归一化效果不明显，有无归一化差别不大。

2、非线性回归模型

基于上述对线性回归模型的分析，考虑使用非线性回归模型。

决策树回归

决策树回归跟决策树分类类似，都是在决定每个结点的划分属性，划分的过程就是建树的过程。决策树回归是将特征空间进行划分，划分的边界平行于坐标轴，每划分一次，随即确定划分单元对应的输出，也就多了一个结点。当根据停止条件划分终止的时候，最终每个单元的输出也就确定了，也就是叶结点。对于测试数据，我们只要按照特征将其归到某个划分单元，便得到对应的输出值。

虽然划分点的选择还是用的最小二乘法，但是测试结果比同样使用最小二乘法的线性回归好很多，MSLE 的值为1.9左右，明显优于线性回归模型。可见上述线性回归模型效果差的主要问题是在于特征和标签间的关系复杂，不能简单当成线性关系。

```
from sklearn import tree
model = tree.DecisionTreeRegressor()
model.fit(x, y)
predicts = model.predict(test_x)
```

同时了解到损失函数可以改为 MAE，而 MAE 对异常点有更好的鲁棒性。该问题查询结果的取值范围较大，预测结果与正确结果的差值范围就较大，哪怕预测只出现少数错误，错误差值较大的话，就会对 MSE 的值产生很大的影响，而这样的错误对 MAE 的影响没有那么大。所以将损失函数由 MSE 改为 MAE 进行了测试，发现 MSLE 的结果为1.7左右，略好于以 MSE 为损失函数的模型。

$$MAE = \frac{\sum |y_{true} - y_{pred}|}{n}$$

随机森林回归

了解到在集成学习类的算法中，随机森林回归算法可以应对特征与标签间存在非线性或复杂关系的情况，而且比其他常见且流行的算法更适合回归问题，因此构建了随机森林模型进行训练。

随机森林属于Bagging类算法，大致思路是训练多个弱模型打包起来组成一个强模型，而强模型的性能要比单个弱模型好很多。而随机森林就是由多棵回归树构成，且森林中的每一棵决策树之间没有关联，模型的最终输出由森林中的每一棵决策树共同决定，所以对训练集中的噪声不敏感，更利于得到一个稳健的模型。

而随机森林的随机性主要体现在两个方面。第一个方面就是样本的随机性，在训练过程中会从训练集中随机抽取一定数量的样本，作为每棵回归树的根结点样本。第二个方面就是特征的随机性，在构建每棵回归树的过程中，随机抽取一定数量的候选特征，从中选择最合适的特征作为分裂结点。

在测试的时候，每一棵回归树的最终预测结果为该样本点到叶结点的均值，而随机森林最终的预测结果是所有回归树预测结果的均值。

因为在上述所分析的决策树回归的测试中得到的结果较好，所以预计在随机森林回归中会有更好的结果。但是测试发现得到的 MSLE 的分数在2.3左右，并没有优于决策树回归。分析可能是随机带来的影响。

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(x, y)
predicts = model.predict(test_x)
```

结果分析

非线性回归模型的结果明显优于线性回归模型，但是 MSLE 分数基本在 2 上下，由于这些模型的损失函数无法设置为 MSLE，所以只能通过调参得到略微改进。考虑后续优化主要有三个方向，一是重新搭建上述非线性回归模型，将 MSLE 设置为损失函数；二是对特征提取进行优化，同时加入对查询计划的处理；三是构建新的回归模型。

3、神经网络模型

在对各优化方向和模型特点进行权衡分析后，我们决定在论文 Learned cardinalities: Estimating correlated joins with deep learning 构建的模型的基础上进行调整，结合之前的模型构建出测试效果好的回归模型。

模型构建与调整

对原有模型的调整主要是在数据处理与损失函数上。

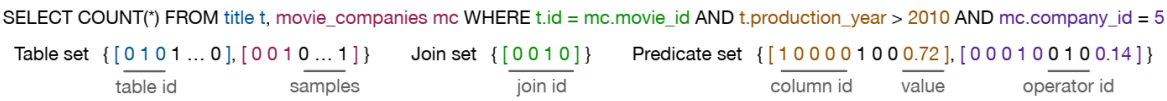
在数据处理上，我们利用上述已完成数据处理部分代替了模型原有的数据处理，在得到的特征向量上进行处理，构造出神经网络模型所需要的数据形式。主要是根据下述编号将得到的特征向量对应不同的部分拆分开，所选择的表、连接条件和属性选择条件分别对应为 table set、join set、predicate set。

```
# 给每个属性一个编号
feature_index = {}
for i in range(len(column_details)):
    feature_index[column_details.iloc[i]['name']] = i

table_index = {'t' : 0, 'mc' : 1, 'ci' : 2, 'mi' : 3, 'mi_idx' : 4, 'mk' : 5}

joins = {'t.id=mi.movie_id', 't.id=mk.movie_id', 't.id=ci.movie_id',
't.id=mc.movie_id', 't.id=mi_idx.movie_id'}
```

得到的特征数据示例为：



而在损失函数部分，原模型的损失函数为，再将模型中的损失函数改为 MSLE

```
loss = tf.keras.losses.MeanSquaredLogarithmicError()
```

而模型的主要部分是直接基于模型原有的形式。该网络为多集卷积网络模型，是一个用于对集合进行操作的神经网络模块，通过全连接多层神经网络得到结果。

网络参数的设置示例：

```
self.sample_mlp1 = nn.Linear(sample_feats, hid_units)
self.sample_mlp2 = nn.Linear(hid_units, hid_units)
```

网络的前向计算示例：

```
hid_sample = F.relu(self.sample_mlp1(samples))
hid_sample = F.relu(self.sample_mlp2(hid_sample))
hid_sample = hid_sample * sample_mask
hid_sample = torch.sum(hid_sample, dim = 1, keepdim = False)
sample_norm = sample_mask.sum(1, keepdim = False)
hid_sample = hid_sample / sample_norm

. . . . . .

#预测结果
out = torch.sigmoid(self.out_mlp2(hid))
return out
```

结果分析

初次测试将 epoch 设置为了100，在配置1070Ti的GPU平台上进行训练，训练时间较长，最终得到的预测结果的MSLE 分数在1.3左右。对epoch、batch size进行调整，再尝试将该模型预测结果与上述非线性回归模型中决策树回归得到的结果进行加权平均，经过多组数据测试，得到最好的的 MSLE 分数值降到1左右。

四、总结

本次实验过程中，我们首先对问题及数据进行了分析，明确了该问题为回归问题，并且在了解了数据集的特点后根据模型所需数据形式对数据进行了处理和特征提取。在构建回归模型的时候，我们先后尝试了线性回归模型和非线性回归模型，并根据模型特点进行了选择与分析，最终发现该数据的特征与预测结果不满足线性关系，且由决策树模型和随机森林模型可以得到较好的结果。在已有模型的优化遇到瓶颈后，我们在论文 Learned cardinalities: Estimating correlated joins with deep learning 中的模型的基础上调整出了效果较好的神经网络模型，并在最终将神经网络模型的预测结果与决策树模型的结果进行加权平均，最终得到 MSLE 分数接近1的测试结果。