哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

大数据分析

实验一

(2019 年度春季学期)

姓　　名　朱明彦
学　　号　1160300314
学　　院　计算机学院
教　　师　杨东华、王金宝

计算机科学与技术学院

目录

实验一 数据预处理

第 1 章 实验目的

掌握数据预处理的步骤和方法，包括数据抽样、数据过滤、数据标准化和归一化、数据清洗。理解数据预处理在各个步骤在大数据环境下的实现方式。

第 2 章 实验环境

- Ubuntu 16.04.5
- Hadoop 2.7.7 伪分布配置
- Spark 2.4.0

第 3 章 实验过程及结果

3.1 数据抽样

输入 数据样式如下，共 4783614 条数据，分层抽样比例 1%

```
144552912|9.349849|56.740876|17.052772|2011/06/27|18.5℃|83.91|38267|1974-06-08|Switzerland|programmer|5042
144552912|9.350188|56.740679|17.614840|2016-10-08|37.8 |78.80|1205|1991-04-14|Italy|teacher|1705
144552912|9.350549|56.740544|18.083536|2010/05/19|14.0℃|80.73|28343|May 25,1989|Luxembourg|farmer|3208
144552912|9.350806|56.740485|18.279465|2014/10/19|2.6℃|80.52|36251|August 25,1992|Belgium|programmer|2565
144552912|9.351053|56.740486|18.422974|2017/12/10|1.2℃|77.90|27133|1992/03/25|Belgium|doctor|2455
144552912|9.351475|56.740502|19.124889|2018-12-01|26.2 |82.83|25448|July 21,1997|France|Manager|2943
144552912|9.352127|56.740558|19.590593|April 18,2010|38.2 |82.93|34087|1991/01/23|Germany|Manager|2984
144552912|9.352420|56.740597|19.621764|2016-01-09|-0.8℃|83.51|18577|April 21,1980|Denmark|programmer|3951
144552912|9.352584|56.740629|19.659931|2014-12-15|29.2 |79.92|25939|1972/11/28|Holland|farmer|2208
144552912|9.352726|56.740663|19.490670|2013/11/05|-8.1℃|77.75|15206|1987-12-22|Italy|accountant|1788
```

输出 数据样式未变，如上面所示；抽样后的结果共有 47737 条数据。

抽样方法 首先统计用于分层变量的 user_carrer 所有可能的取值，并将所有的职业对应的分层比均置为 1%，利用分层抽样的思想即可抽得上述样本，具体如下代码所示。

```
1 def parse(x):
2     return x.split("|")
3
4 origin_record = sc.textFile("hdfs://.../input/large_data.txt").map(parse)
5 sample_record = origin_record.map(lambda x: (x[10], x))\
```

```
    .sampleByKey(False, dic, seed=seed).map(lambda x: x[1])
sample_record.map(lambda x: "|".join(x)).coalesce(1).saveAsTextFile("D_Sample")
```

3.2 数据过滤

输入 有两个部分，其一为抽样结果 D_Sample，共 47737 条数据；其二为原始数据即 D，共 4783614 条数据。

输出 过滤筛选掉 longitude 不在 $[8.1461259, 11.1993265]$，latitude 不在 $[56.5824856, 57.750511]$ 以及 rating 在前 1% 和后 1% 的无效数据，得到的结果共 4689245 条数据，即 D_Filtered。

过滤方法 将抽样数据 D_Sample 按照 rating 进行排序，并提取前 1% 以及后 1% 对应的临界值，将提取后的临界值与经纬度坐标的范围同时作为过滤条件，对原始数据 D 进行过滤即可，具体见代码。

```
def filter_func(x):
    lng = float(x[1])
    lat = float(x[2])
    ans1 = longitude[0] <= lng <= longitude[1]
    ans2 = latitude[0] <= lat <= latitude[1]
    ans3 = x[6] == '?' or (limit[0] <= float(x[6]) <= limit[1])
    return ans1 and ans2 and ans3

filtered_record_temp = sample_record.sortBy(lambda x: x[6]).collect()
length = len(filtered_record_temp)
outliers = length // 100
limit = float(filtered_record_temp[outliers][6]),
            float(filtered_record_temp[-outliers][6])
filtered_record = origin_record.filter(filter_func)
filtered_record.map(lambda x: "|".join(x)).coalesce(1).saveAsTextFile("D_filtered")
```

3.3 数据标准化和归一化

输入 经过过滤后的数据 D_Filtered，数据格式如下

```
144552912|9.349849|56.740876|17.052772|2011/06/27|18.5℃|83.91|38267|1974-06-08|Switzerland|programmer|5042
144552912|9.350188|56.740679|17.614840|2016-10-08|37.8 |78.80|1205|1991-04-14|Italy|teacher|1705
144552912|9.350549|56.740544|18.083536|2010/05/19|14.0℃|80.73|28343|May 25,1989|Luxembourg|farmer|3208
144552912|9.350806|56.740485|18.279465|2014/10/19|2.6℃|80.52|36251|August 25,1992|Belgium|programmer|2565
```

```
144552912|9.351053|56.740486|18.422974|2017/12/10|1.2℃|77.90|27133|1992/03/25|Belgium|doctor|2455
144552912|9.351475|56.740502|19.124889|2018-12-01|26.2 |82.83|25448|July 21,1997|France|Manager|2943
144552912|9.352127|56.740558|19.590593|April 18,2010|38.2 |82.93|34087|1991/01/23|Germany|Manager|2984
144552912|9.352420|56.740597|19.621764|2016-01-09|-0.8℃|83.51|18577|April 21,1980|Denmark|programmer|3951
144552912|9.352584|56.740629|19.659931|2014-12-15|29.2 |79.92|25939|1972/11/28|Holland|farmer|2208
144552912|9.352726|56.740663|19.490670|2013/11/05|-8.1℃|77.75|15206|1987-12-22|Italy|accountant|1788
```

输出 将数据中温度的单位统一为 ℃，并且统一所有日期的格式为 YYYY-MM-DD，将 rating 数据归一化，最终结果数据格式如下

```
144552912|9.349849|56.740876|17.052772|2011-06-27|18.5℃|0.67|38267|1974-06-08|Switzerland|programmer|5042
144552912|9.350188|56.740679|17.614840|2016-10-08|20.0℃|0.53|1205|1991-04-14|Italy|teacher|1705
144552912|9.350549|56.740544|18.083536|2010-05-19|14.0℃|0.58|28343|1989-05-25|Luxembourg|farmer|3208
144552912|9.350806|56.740485|18.279465|2014-10-19|2.6℃|0.57|36251|1992-08-25|Belgium|programmer|2565
144552912|9.351053|56.740486|18.422974|2017-12-10|1.2℃|0.50|27133|1992-03-25|Belgium|doctor|2455
144552912|9.351475|56.740502|19.124889|2018-12-01|8.4℃|0.64|25448|1997-07-21|France|Manager|2943
144552912|9.352127|56.740558|19.590593|2010-04-18|20.4℃|0.64|34087|1991-01-23|Germany|Manager|2984
144552912|9.352420|56.740597|19.621764|2016-01-09|-0.8℃|0.66|18577|1980-04-21|Denmark|programmer|3951
144552912|9.352584|56.740629|19.659931|2014-12-15|11.4℃|0.56|25939|1972-11-28|Holland|farmer|2208
144552912|9.352726|56.740663|19.490670|2013-11-05|-8.1℃|0.50|15206|1987-12-22|Italy|accountant|1788
```

数据标准化和归一化方法 简单利用规则进行 Map 即可，注意华氏度和摄氏度的转化方法以及日期的不同格式，具体见代码如下，其中第 29 行判断的是华氏度符号是否存在，此处显示有误。

```
1  def time_temp_parse(x):
2      standard_time_format = "%Y-%m-%d"
3      unstandard_format = "%Y/%m/%d"
4      unstandard_format_2 = "%B %d,%Y"
5      review_date = x[4]
6
7      temperature = x[5]
8      review_date_time = None
9      if '/' in review_date:
10         review_date_time = datetime.strptime(review_date, unstandard_format)
11         review_date = review_date_time.strftime(standard_time_format)
12     elif ',' in review_date:
13         review_date_time = datetime.strptime(review_date, unstandard_format_2)
14         review_date = review_date_time.strftime(standard_time_format)
15     x[4] = review_date
16
17     user_birthday = x[8]
```

```
    user_birthday_time = None
    if '/' in user_birthday:
        user_birthday_time = datetime.strptime(
            user_birthday, unstandard_format)
        user_birthday = user_birthday_time.strftime(standard_time_format)
    elif ',' in user_birthday:
        user_birthday_time = datetime.strptime(
            user_birthday, unstandard_format_2)
        user_birthday = user_birthday_time.strftime(standard_time_format)
    x[8] = user_birthday

    if ' ' in temperature:
        temperature = "%.1f" % (float(temperature[:-1]) - 32 / 1.8) + "℃"
        x[5] = temperature

    rating = "%.2f" % ((float(x[6]) - limit[0]) / (limit[1] - limit[0]))\
                        if x[6] != "?" else x[6]
    x[6] = rating
    return x
normalization_record = filtered_record.map(time_temp_parse)
```

3.4 数据清洗

输入 经过数据标准化和归一化之后的结果，经过统计共缺少 9444 个不同的信息，查询的方式如下面的 Shell 命令。

```
cat D_Filtered/part-* | grep "?" | wc -l
```

输出 将所有的缺失信息进行填充之后的结果，即 D_Preprocessed；同样执行上述的 Shell 命令，得到的结果为 0 条信息。

数据填充方法 主要分为以下两步，分别用于填充 user_income 和 rating。

1. 根据 user_career 和 user_nationlity 进行分组，将每个分组的薪资 user_income 平均值作为具有同样国籍和职业但缺失薪资的元组的填充值。

2. 将根据第一步填充好的结果，利用机器学习中线性回归的方式和未缺失 rating 的数据中 longitude，latitude，altitude，user_income 学习到相关的模型，对相应缺失值进行预测。

具体的代码如下所示。

```
def fill_rating(x):
    if x[6] != "?":
        return x
    else:
        _features = np.array([float(x[1]), float(x[2]), float(x[3]), float(x[-1])])
        return x[:6] + ["%.2f" % (_features.dot(coefficients))] + x[7:]

spark = SparkSession(sc)
training_data = spark.createDataFrame(fill_missing_income.filter(lambda x: x[6] != "?")\
                .map(lambda x: (float(x[6]), DenseVector([float(x[1]), float(x[2]),
                float(x[3]), float(x[-1])]))), ["label", "features"])
lr = LinearRegression(maxIter=10, regParam=0.2, elasticNetParam=0, fitIntercept=False)
lrModel = lr.fit(training_data)
coefficients = np.array(lrModel.coefficients)
fill_missing = fill_missing_income.map(fill_rating)
fill_missing.map(lambda x: "|".join(x)).coalesce(1).saveAsTextFile("D_Preprocessed")
```

第 4 章 实验心得

实验环境在上学期相应的课程中已经完成了搭建，所以在搭建环境方面没有耗费很多时间。在最后进行数据填充时，开始使用的是 Spark 在 0.x 版本时期提出的 MLlib 库，出现了很多莫名奇妙的问题，比如过拟合无法通过增加正则项减轻，即使使用更大的惩罚项的参数，迭代更多次，效果依然不好；后来由于这是被废弃的 API，所以换用回 Spark 在 2.x 时期的 ML 库中的相关函数，问题就得到了解决。