

# 一、Spark配置

## 1. 单机模式

### 1.1 下载Anaconda3

下载 Anaconda3-2021.11-Linux-x86\_64.sh:

```
wget https://repo.anaconda.com/archive/Anaconda3-2021.11-Linux-x86_64.sh
```

解压安装包Anaconda3-5.1.0-Linux-x86\_64.sh至路径 /home/Lsc:

```
sudo tar -zxvf ~/Anaconda3-2021.11-Linux-x86_64.sh -C /home/Lsc
```

安装Anaconda3:

```
sh ./Anaconda3-2021.11-Linux-x86_64.sh
```

接着然后一直回车然后空格，等出现接受许可界面，输入 yes 接收许可。

### 1.2 配置文件

配置国内源:

```
sudo vim ~/.condarc
```

追加以下内容:

```
channels:
  - defaults
show_channel_urls: true
default_channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2
custom_channels:
  conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  msys2: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  bioconda: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  menpo: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  simpleitk: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
```

创建pyspark环境:

```
conda create -n pyspark python=3.6 # 基于python3.6创建pyspark虚拟环境
conda activate pyspark # 激活（切换）到pyspark虚拟环境
```

在pyspark环境中使用pip下载pyhive、pyspark、jieba包:

```
pip install pyspark==2.4.0 jieba pyhive -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

### 1.3 Spark Local模式搭建

下载 spark-2.4.8，解压安装包spark-2.4.8-bin-without-hadoop至路径 /home/Lsc:

```
sudo tar -zxvf ~/spark-2.4.8-bin-without-hadoop -C /home/Lsc
```

重命名:

```
mv spark-2.4.8-bin-without-hadoop/ spark-2.4.8
```

配置环境变量:

vi ~/.bashrc 修改 .bashrc 文件，在文件末尾追加:

```
# 默认启动pyspark虚拟环境
conda activate pyspark

# JAVA_HOME
export JAVA_HOME=/home/Lsc/jdk1.8.0_381
# PYSARK_PYTHON
export PYSARK_PYTHON=/home/Lsc/anaconda3/envs/pyspark/bin/python
export PATH=$JAVA_HOME/bin:$PATH
export SPARK_HOME=/home/Lsc/spark
export PATH=$SPARK_HOME/bin:$PATH
```

配置 /etc/profile，文件末尾追加:

```
export SPARK_HOME=/home/Lsc/spark-2.4.8
# HADOOP_CONF_DIR
export HADOOP_CONF_DIR=$HADOOP_HOME/home/Lsc/hadoop-3.3.6
# PYSARK_PYTHON
export PYSARK_PYTHON=/home/Lsc/anaconda3/envs/pyspark/bin/python
export
PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$SPARK_HOME/bin:$PATH
```

使环境变量文件立即生效:

```
source /etc/profile
source ~/.bashrc
```

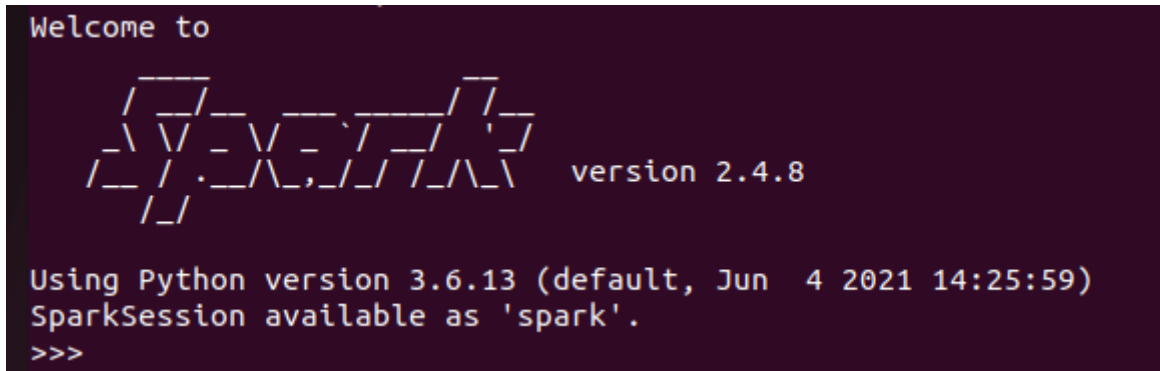
配置spark-env.sh:

```
cd /opt/spark-2.4.0/conf
cp spark-env.sh.template spark-env.sh
```

在该文件最后追加以下内容:

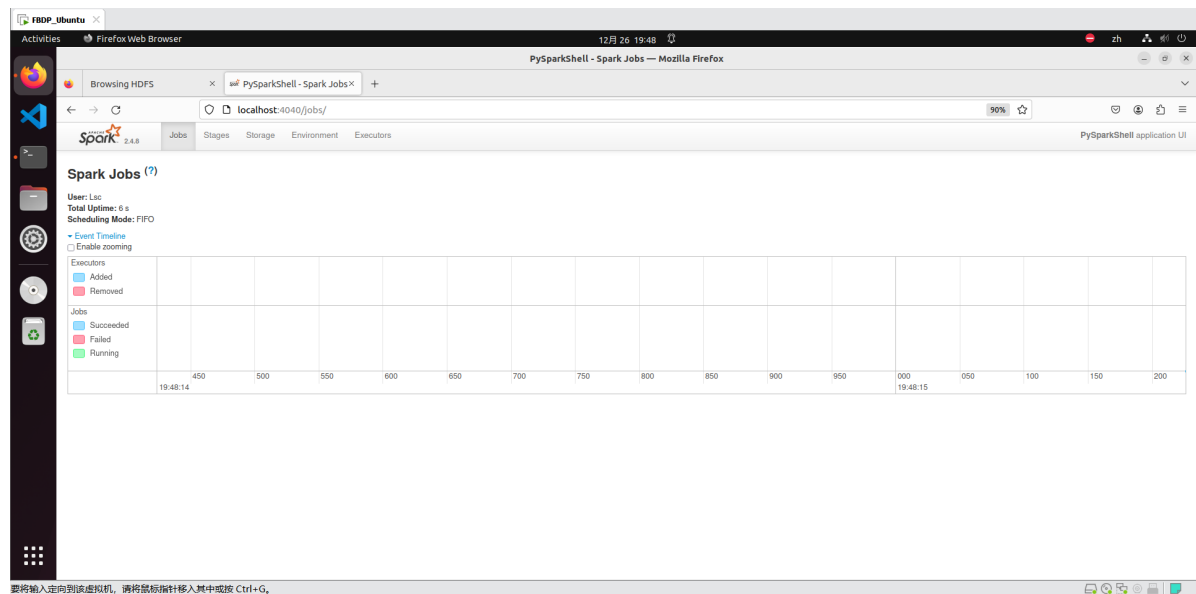
```
export JAVA_HOME=/home/Lsc/jdk1.8.0_381
export SPARK_DIST_CLASSPATH=$(/home/Lsc/hadoop-3.3.6/bin/hadoop classpath)
```

验证Spark是否安装成功：



可见spark可以正常开启

WebUI截图：



## 1.4 配置Jupyter Notebook

Jupyter notebook是一个很好的数据交互式工具，以下是具体配置步骤：

### 1. 激活 pyspark 环境：

打开终端，运行以下命令来激活 pyspark 环境：

```
conda activate pyspark
```

### 2. 安装Jupyter：

```
conda install jupyter
```

运行 `jupyter notebook --version` 判断是否安装成功

### 3. 配置jupyter notebook根目录：

安装完Jupyter运行后发现根目录为一个陌生的目录，并不方便使用，需要配置jupyter notebook每次打开的根目录，在激活 pyspark 的环境下运行如下命令：

```
jupyter notebook --generate-config
```

生成配置文件，去给出的文件地址：

```
/home/Lsc/.jupyter\jupyter_notebook_config.py
```

打开jupyter\_notebook\_config.py文件，修改配置文件：

将 `#c.NotebookApp.notebook_dir = ''` 改为  
`c.NotebookApp.notebook_dir = '/home/Lsc'`

默认情况下，Jupyter Notebook 只能从启动服务器的机器上访问，通过如下设置启用远程访问：

```
pythonCopy codec.NotebookApp.ip = '0.0.0.0' # 允许任何IP访问
c.NotebookApp.open_browser = False # 不自动打开浏览器
```

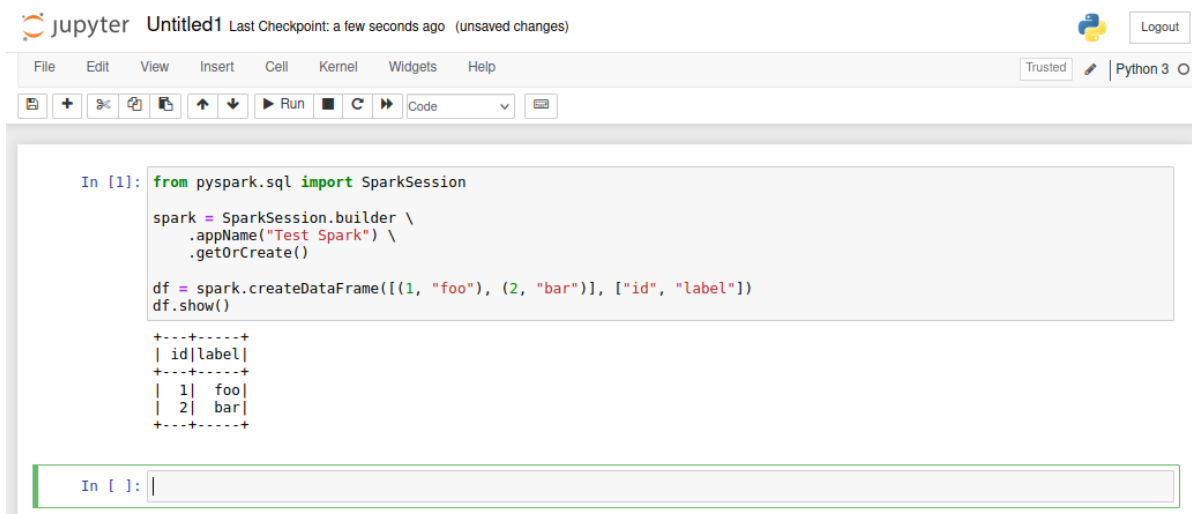
#### 4. 运行jupyter notebook：

在pyspark环境下执行 jupyter notebook 开启：



#### 5. 检验jupyter notebook是否连接spark：

创建一个新的Notebook，选择pyspark环境的内核，运行Spark测试代码以确认一切正常。这段代码创建了一个简单的 Spark DataFrame，包含两列（id 和 label）和两行数据，然后使用 show() 方法将其内容打印出来：



可见PySpark环境中测试代码输出结果正常，Spark环境正确配置，且jupyter 已成功连接spark。

## 二、银行贷款违约预测任务

## 1. 任务一

1.1 编写 Spark 程序，统计application\_data.csv中所有用户的贷款金额AMT\_CREDIT 的分布情况，以 10000 元为区间进行输出：

python代码如下：

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, floor

# 创建 SparkSession
spark = SparkSession.builder.appName("LoanAmountDistribution").getOrCreate()

# 读取 CSV 文件
df = spark.read.csv("file:///home/Lsc/application_data.csv", header=True,
inferSchema=True)

# 选择并处理贷款金额列
df = df.withColumn("AMT_CREDIT", col("AMT_CREDIT").cast("double")) # 确保列为浮点
型
df = df.withColumn("Credit_Range", floor(col("AMT_CREDIT") / 10000) * 10000)

# 计算每个区间的贷款金额分布
distribution = df.groupBy("Credit_Range").count()

# 格式化输出
formatted_output = distribution.rdd.map(lambda x: ((x[0], x[0] + 10000), x[1]))
# 对结果进行排序
sorted_output = formatted_output.sortBy(lambda x: x[0])

# 重分区到一个分区
single_partition_rdd = sorted_output.coalesce(1)
sorted_output.collect()
# 保存到文本文件
output_path = "file:///home/Lsc/FBDP_homework/homework4/output1-1"
single_partition_rdd.saveAsTextFile(output_path)

# 重分区到一个分区
single_partition_rdd = sorted_output.coalesce(1)

# 停止 Spark Session
spark.stop()
```

输出结果保存在文件 output1-1 之中，下面给出部分结果：

```

1 ((40000, 50000), 561)
2 ((50000, 60000), 891)
3 ((60000, 70000), 719)
4 ((70000, 80000), 1226)
5 ((80000, 90000), 668)
6 ((90000, 100000), 1939)
7 ((100000, 110000), 1871)
8 ((110000, 120000), 1930)
9 ((120000, 130000), 1323)
10 ((130000, 140000), 4792)
11 ((140000, 150000), 2239)
12 ((150000, 160000), 3653)
13 ((160000, 170000), 1919)
14 ((170000, 180000), 2131)

```

## 1.2 编写Spark程序，统计application\_data.csv中客户贷款金额AMT\_CREDIT比客户收入AMT\_INCOME\_TOTAL差值最高和最低的各十条记录：

python代码如下：

```

from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# 初始化Spark会话
spark = SparkSession.builder.appName("LoanIncomeDifference").getOrCreate()

# 读取 CSV 文件
df = spark.read.csv("file:///home/Lsc/application_data.csv", header=True,
inferSchema=True)

# 计算差值
df = df.withColumn("Difference", col("AMT_CREDIT") - col("AMT_INCOME_TOTAL"))

# 获取差值最高的十条记录
top_10_max_diff = df.orderBy(col("Difference").desc()).limit(10)

# 获取差值最低的十条记录
top_10_min_diff = df.orderBy(col("Difference")).limit(10)

# 输出结果
top_10_max_diff.select("SK_ID_CURR", "NAME_CONTRACT_TYPE", "AMT_CREDIT",
"AMT_INCOME_TOTAL", "Difference").show()
top_10_min_diff.select("SK_ID_CURR", "NAME_CONTRACT_TYPE", "AMT_CREDIT",
"AMT_INCOME_TOTAL", "Difference").show()

# 关闭Spark会话
spark.stop()

```

## 客户贷款金额AMT\_CREDIT比客户收入AMT\_INCOME\_TOTAL差值最高的十条记录：

```

+-----+-----+-----+-----+-----+
|SK_ID_CURR|NAME_CONTRACT_TYPE|AMT_CREDIT|AMT_INCOME_TOTAL|Difference|
+-----+-----+-----+-----+-----+
|    433294|    Cash loans| 4050000.0|    405000.0| 3645000.0|
|    210956|    Cash loans| 4031032.5|    430650.0| 3600382.5|
|    434170|    Cash loans| 4050000.0|    450000.0| 3600000.0|
|    315893|    Cash loans| 4027680.0|    458550.0| 3569130.0|

```

	238431	Cash loans	3860019.0	292050.0	3567969.0
	240007	Cash loans	4050000.0	587250.0	3462750.0
	117337	Cash loans	4050000.0	760846.5	3289153.5
	120926	Cash loans	4050000.0	783000.0	3267000.0
	117085	Cash loans	3956274.0	749331.0	3206943.0
	228135	Cash loans	4050000.0	864900.0	3185100.0
	+-----+-----+-----+-----+-----+				

客户贷款金额AMT\_CREDIT比客户收入AMT\_INCOME\_TOTAL差值最低的十条记录:

	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_CREDIT	AMT_INCOME_TOTAL	Difference
	114967	Cash loans	562491.0	1.17E8	-1.16437509E8
	336147	Cash loans	675000.0	1.800009E7	-1.732509E7
	385674	Cash loans	1400503.5	1.35E7	-1.20994965E7
	190160	Cash loans	1431531.0	9000000.0	-7568469.0
	252084	Cash loans	790830.0	6750000.0	-5959170.0
	337151	Cash loans	450000.0	4500000.0	-4050000.0
	317748	Cash loans	835380.0	4500000.0	-3664620.0
	310601	Cash loans	675000.0	3950059.5	-3275059.5
	432980	Cash loans	1755000.0	4500000.0	-2745000.0
	157471	Cash loans	953460.0	3600000.0	-2646540.0
	+-----+-----+-----+-----+-----+				

## 2. 任务二

2.1 统计所有男性客户 (CODE\_GENDER=M) 的小孩个数 (CNT\_CHILDREN) 类型占比情况:

python代码如下:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, count, format_number

# 初始化Spark会话
spark = SparkSession.builder.appName("GenderChildrenAnalysis").getOrCreate()

# 读取 CSV 文件
df = spark.read.csv("file:///home/Lsc/application_data.csv", header=True,
inferSchema=True)

# 过滤出男性客户
men_df = df.filter(df["CODE_GENDER"] == "M")
# 计算男性客户总数
total_men_count = men_df.count()

# 对 CNT_CHILDREN 进行分组并计算每组的数量
result = men_df.groupBy("CNT_CHILDREN").count()

# 计算每种小孩个数的占比并格式化为保留8位小数
result = result.withColumn("formatted_ratio", format_number((col("count") /
total_men_count), 8))

# 选择需要的列
result_df = result.select("CNT_CHILDREN", "count", "formatted_ratio")

# 按照 CNT_CHILDREN 升序排列
```

```
ordered_result_df = result_df.orderBy("CNT_CHILDREN")

# 显示结果
ordered_result_df.show()

# 关闭 SparkSession
spark.stop()
```

输出结果:

```
+-----+-----+-----+
|CNT_CHILDREN|count|formatted_ratio|
+-----+-----+-----+
|          0|70318|      0.66931914|
|          1|22660|      0.21568833|
|          2|10413|      0.09911573|
|          3| 1446|      0.01376369|
|          4|  170|      0.00161814|
|          5|   33|      0.00031411|
|          6|   11|      0.00010470|
|          7|    4|      0.00003807|
|          8|    1|      0.00000952|
|          9|    1|      0.00000952|
|         11|    1|      0.00000952|
|         14|    1|      0.00000952|
+-----+-----+-----+
```

**2.2 统计每个客户出生以来每天的平均收入 (avg\_income) =总收入 (AMT\_INCOME\_TOTAL) /出生天数 (DAYS\_BIRTH), 统计每日收入大于1的客户, 并按照从大到小排序, 保存为csv:**

python代码如下:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import abs, col, format_number

# 初始化Spark会话
spark = SparkSession.builder.appName("AverageDailyIncome").getOrCreate()

# 读取 CSV 文件
df = spark.read.csv("file:///home/Lsc/application_data.csv", header=True,
inferSchema=True)

# 计算每天的平均收入, 并使用format_number保留五位小数, 注意DAYS_BIRTH是负值, 需要取绝对值
#df = df.withColumn("avg_income", format_number(col("AMT_INCOME_TOTAL") /
abs(col("DAYS_BIRTH")), 5))
df = df.withColumn("DAYS_BIRTH_ABS", abs(col("DAYS_BIRTH")))
df = df.withColumn("avg_income", col("AMT_INCOME_TOTAL") /
col("DAYS_BIRTH_ABS"))
# 筛选出每日收入大于1的客户
filtered_df = df.filter(col("avg_income") > 1)
# 按照每日平均收入降序排序
sorted_df = filtered_df.orderBy(col("avg_income").desc())

# 选择需要的列
df_final = sorted_df.select("SK_ID_CURR", "avg_income")

# 将结果汇总到一个csv文件中
```



```
df_final.coalesce(1).write.csv("file:///home/Lsc/22", header=True)
```

```
# 关闭Spark会话  
spark.stop()
```

输出结果:

	Standard	Standard
1	SK_ID_CURR	avg_income
2	114967	9274.673008323425
3	336147	1146.2105196128375
4	385674	996.2364401151207
5	190160	547.945205479452
6	219563	417.51716459454445
7	310601	373.63408059023834
8	157471	360.4325190228274

### 3. 任务三

根据给定的数据集，基于Spark MLlib 或者Spark ML编写程序对贷款是否违约进行分类，并评估实验结果的准确率。该任务可视为一个“二分类”任务，因为数据集只存在两种情况，违约（Class=1）和其他（Class=0）。

按照8: 2的比例将数据集application\_data.csv随机拆分成训练集和测试集。最后评估模型的性能，评估指标为accuracy、f1-score。

这部分内容写在了jupyter notebook内，详情见文件

#### 1. MLP (多层感知器)

- 准确率 (Accuracy): 0.9143
- 召回率 (Recall): 0.8915
- F1 分数 (F1 Score): 0.9027

#### 2. GBT 分类器 (Gradient Boosting Trees Classifier)

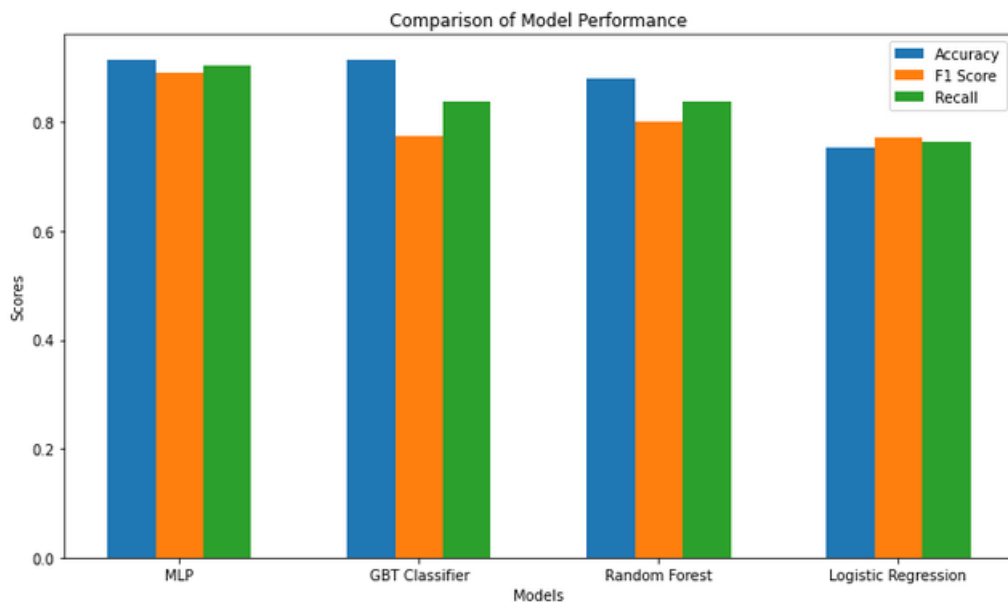
- 准确率 (Accuracy): 0.9152
- 召回率 (Recall): 0.7737
- F1 分数 (F1 Score): 0.8385

#### 3. 随机森林 (Random Forest)

- 准确率 (Accuracy): 0.8793
- 召回率 (Recall): 0.7999
- F1 分数 (F1 Score): 0.8377

#### 4. 逻辑回归 (Logistic Regression)

- 准确率 (Accuracy): 0.7531
- 召回率 (Recall): 0.7726
- F1 分数 (F1 Score): 0.7627



### 三、遇到的问题

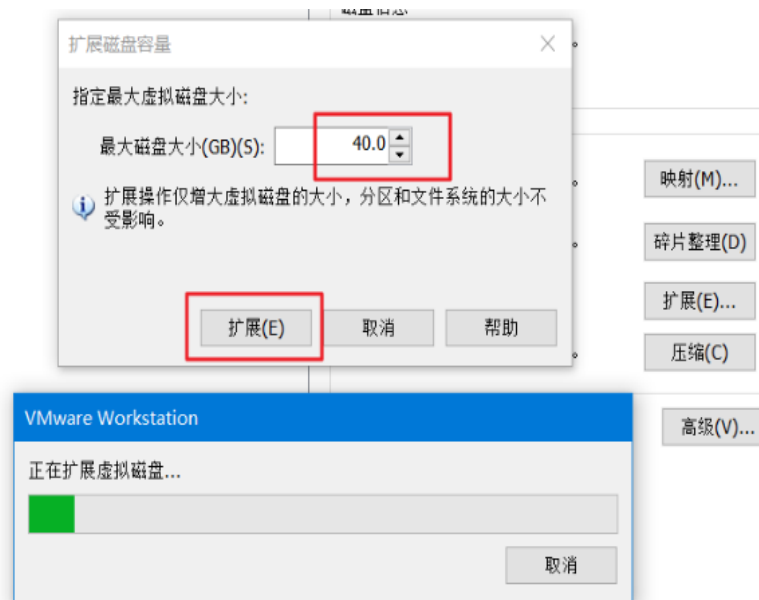
#### 1.虚拟机磁盘空间不足

- 点击编辑虚拟机设置-> 硬盘-> 扩展（此处如果扩展不可点，需要删除所有快照）：





- 弹出页面填写要增加到多少内存->扩展，等待完成即可：



- 开机后进入recover模式，删除日志文件，腾出空间，以便可以成功安装gparted：
- 点击**开启虚拟机**时，按住**shift** 键，用方向键选择 [ubuntu] (<https://so.csdn.net/so/search?q=ubuntu&spm=1001.2101.3001.7020>) 高级选项 -> **回车**进入 -> 选择 **recover mode** -> **回车**进入：

```
df -h          # 查看磁盘使用情况，可看到 sda1 使用率为 100%
cd /var/log/cups # 进入文件夹
ls -lhs       # 将文件以从大到小顺序展现

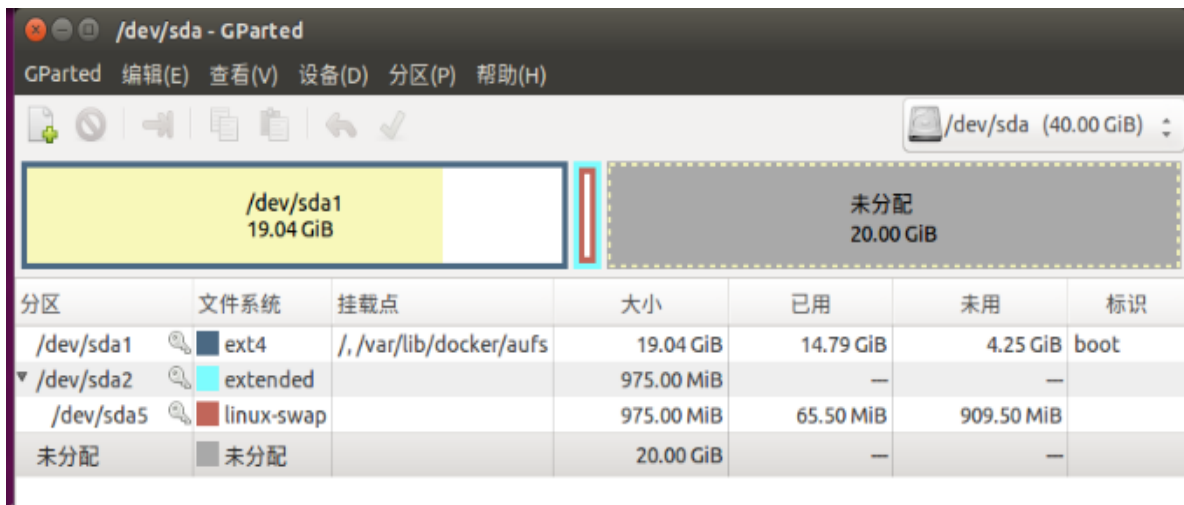
rm -rf error_log # 删除没用的日志文件
ls -lhs         # 再次查看
df -h          # 再次查看使用率为93%

reboot         # 重启进入系统
```

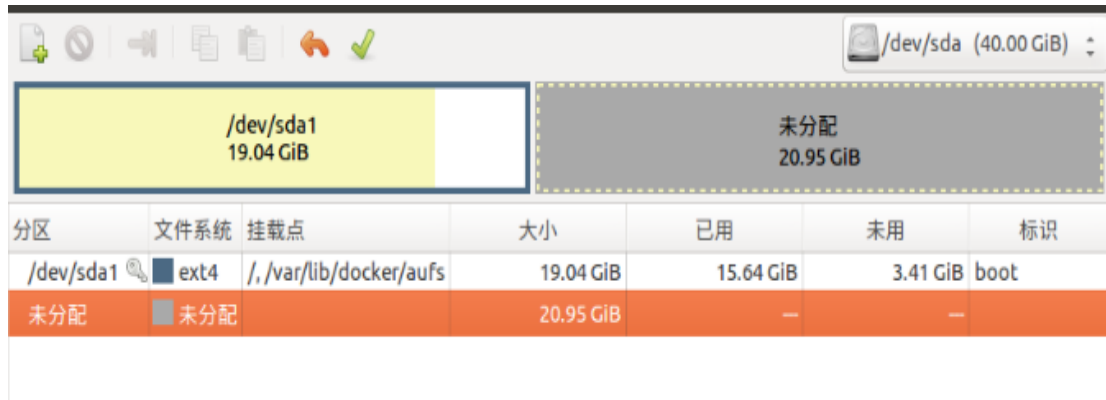
```
root@hl-PC:~#  
root@hl-PC:~#  
root@hl-PC:~# df -h  
Filesystem      Size  Used Avail Use% Mounted on  
udev            956M   0  956M   0% /dev  
tmpfs           198M  3.4M  194M   2% /run  
/dev/sda1       19G   19G    0 100% /  
tmpfs           986M   0  986M   0% /dev/shm  
tmpfs           5.0M  4.0K  5.0M   1% /run/lock  
tmpfs           986M   0  986M   0% /sys/fs/cgroup  
root@hl-PC:~# cd /var/log/cups  
root@hl-PC:/var/log/cups# ls -l  
total 7.7G  
-rw-r----- 1 root adm 6.3G May 30 20:26 error_log  
-rw-r----- 1 root adm 1.5G May 30 18:23 error_log.1  
-rw-r----- 1 root adm 2.2K May 30 18:12 access_log.1  
-rw-r----- 1 root adm 277 May 20 17:01 access_log.4.gz  
-rw-r----- 1 root adm 242 Apr 14 07:38 access_log.6.gz  
-rw-r----- 1 root adm 240 May 28 19:11 access_log.2.gz  
-rw-r----- 1 root adm 225 May 18 09:47 access_log.5.gz  
-rw-r----- 1 root adm 200 May 21 07:35 access_log.3.gz  
-rw-r----- 1 root adm 109 May 19 10:14 error_log.2.gz  
-rw-r----- 1 root adm 0 May 30 18:21 access_log  
-rw-r----- 1 root adm 0 Apr 9 09:59 page_log  
root@hl-PC:/var/log/cups#
```

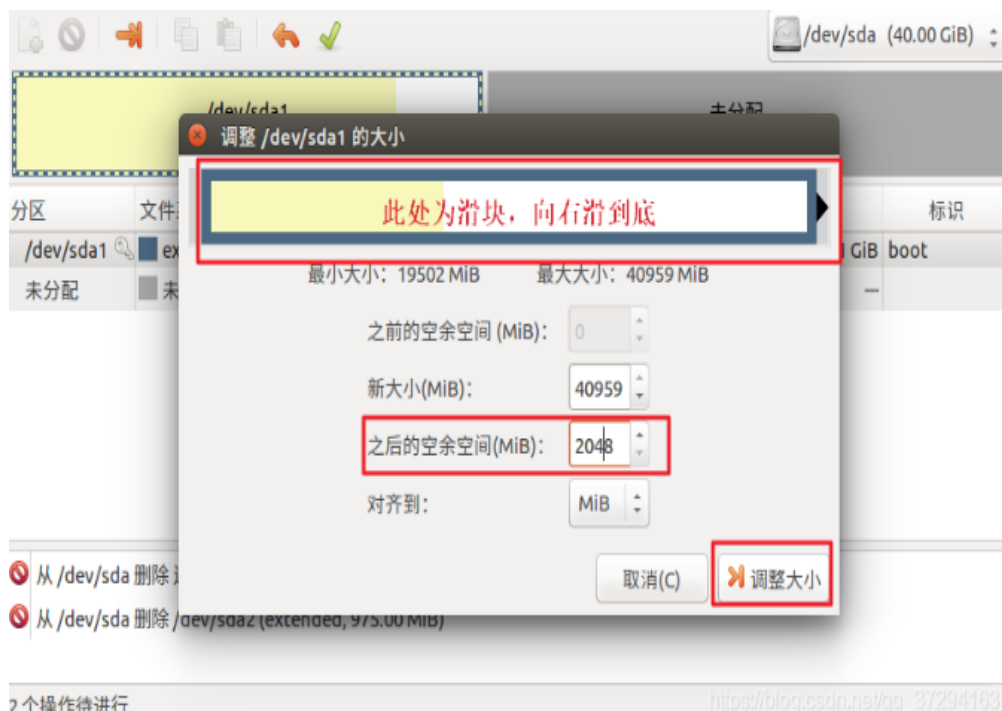
- 安装 gparted, 重新分配内存

```
sudo apt-get install gparted # 安装  
sudo gparted #启动
```



- 重新调整分区:





- 最终成功扩展虚拟机磁盘空间

## 2.无法用conda命令新建python环境

用conda命令 `conda create -n pyspark python=3.6` 新建python环境时，一直报错：

```
CondaHTTPError: HTTP 000 CONNECTION FAILED for url
An HTTP error occurred when trying to retrieve this URL
```

一开始以为是源的问题，更换各种形式的源（比如清华源、中科大源、阿里云等）之后仍报错  
更换https为http仍然无法解决问题，开始反思是不是走了代理：

```
(base) Lsc@Lsc-virtual-machine:~$ export | grep -i prox
declare -x ALL_PROXY="socks://192.168.126.1:7890/"
declare -x FTP_PROXY="http://192.168.126.1:7890/"
declare -x HTTPS_PROXY="http://192.168.126.1:7890/"
declare -x HTTP_PROXY="http://192.168.126.1:7890/"
declare -x NO_PROXY="localhost,127.0.0.0/8,::1"
declare -x all_proxy="socks://192.168.126.1:7890/"
declare -x ftp_proxy="http://192.168.126.1:7890/"
declare -x http_proxy="http://192.168.126.1:7890/"
declare -x https_proxy="http://192.168.126.1:7890/"
declare -x no_proxy="localhost,127.0.0.0/8,::1"
```

为了测试是否是代理设置导致的问题，临时禁用代理设置：

```
unset ALL_PROXY FTP_PROXY HTTPS_PROXY HTTP_PROXY NO_PROXY all_proxy ftp_proxy
http_proxy https_proxy no_proxy
```

然后再次尝试使用 Conda命令，实现成功下载。

分析可能的原因如下：

- 代理服务器配置：

代理配置指向 **192.168.126.1** 端口 **7890**，如果此代理服务器不正确配置、不可访问，或不允许访问 **Conda** 的服务器，则会导致下载失败。

- 不一致的代理类型：

环境变量中设置了两种类型的代理：**HTTP** (<http://192.168.126.1:7890/>) 和 **SOCKS** (<socks://192.168.126.1:7890/>)，这可能导致某些应用或服务无法正确解析或使用这些代理设置。

---

## 四、总结收获

本实验围绕银行贷款违约预测，使用Spark处理和分析约30万条贷款数据。目标是理解和预测贷款违约风险，增进对金融数据分析的认识。下面是我的总结与反思：

### 1. 数据处理和统计分析：

- 学习了如何使用Spark进行基本的数据处理，如贷款金额分布的计算。
- 掌握了关键财务指标的分析方法。

### 2. Spark SQL应用：

- 理解了在大数据环境下使用Spark SQL进行数据查询和分析的方法。
- 掌握了如何统计特定的数据特征。

### 3. 机器学习模型训练：

- 通过构建和评估机器学习模型，增强了对Spark MLlib和Spark ML的理解。
- 学习了如何处理、分析数据，并应用于实际的分类模型中。

### 4. 实验挑战与解决方案：

- 在数据处理过程中遇到的挑战主要集中在数据的清洗和预处理上。
- 通过实验学会了如何针对具体问题调整数据处理策略。

本次实验使用Spark进行数据处理和分析，对银行贷款违约分类任务有了更深刻的理解。后续需要深入学习Spark的高级功能，探索更多机器学习算法在金融风险评估中的应用。