



重庆大学

研究生课程考核试卷

科 目: 大数据架构与技术 教 师: 冯永

姓 名:

学 号:

专 业: 电子信息、电子信息、电子信息、电子信息

类 别: 专硕、专硕、专硕、专硕

上课时间: 2022年9月至2022年11月

考 生 成 绩:

卷面成绩	平时成绩	课程综合成绩

阅卷评语:

阅卷教师 (签名)

重庆大学研究生院制

目录

1 数据集.....	1
1.1 分布式爬虫项目部署.....	1
1.1.1 反爬手段分析.....	1
1.1.2 构造获取评论的 URL.....	2
1.1.3 本地环境设置与测试.....	4
1.1.4 多 PC 环境设置与测试.....	5
1.2 构造数据集.....	6
1.3 数据集格式说明和使用方案.....	6
1.3.1 数据集格式说明.....	6
1.3.2 数据集使用方案.....	7
1.4 小结.....	7
2 环境搭建.....	8
2.1 环境搭建具体流程.....	8
2.1.1 安装 Ubuntu.....	8
2.1.2 安装 JDK	8
2.1.3 安装 hadoop.....	9
2.1.4 修改 hadoop 配置文件	9
2.1.5 安装 spark.....	12
2.1.6 修改 Spark 配置文件	12
2.2 测试用例具体流程.....	13
2.2.1 启动服务.....	13
2.2.2 上传数据集和准备程序.....	14
2.2.3 执行 WordCount 算例	15
2.3 小结.....	16
3 算法.....	18
3.1 准备工作.....	18
3.2 算法原理.....	19
3.2.1 加载数据集.....	19
3.2.2 数据预处理.....	19
3.2.3 特征提取管道.....	20
3.2.4 建立 SVM 模型	21
3.2.5 模型评估.....	22

3.3 结果分析与可视化.....	23
3.4 小结.....	24
4 创新性思路和策略.....	26
5 总结与展望.....	27
5.1 项目带来的收获.....	27
5.2 课程带来的收获.....	27
附录 A：小组分工.....	28

1 数据集

通过本章，我们将通过分布式爬虫采集一份 10 万规模的数据集。本章将具体介绍分布式爬虫的实现原理，最后将介绍我们所收集的数据集的格式说明和使用方案。

1.1 分布式爬虫项目部署

本次课程项目中，我们希望爬取同程旅游网站中重庆地区所有景点的评论和评价，从而能够构造一个数据集便于后续任务即大数据分析相关算法的使用。我们将基于 Scrapy-Redis 框架来设计整个爬虫项目。

1.1.1 反爬手段分析

① 同程网的反爬手段

我们登入同程网，尝试快速访问多个页面。随后我们发现访问页面过多会出现滑块验证。如图 1.1 所示。因此，如果我们使用传统爬虫方法就会遭遇困难，传统方法如爬取所有 HTML 页面并通过标签选择来获取评论信息和评价信息。



图 1.1 同程网的反爬手段
Figure1.1: Anti-spider measures

② 解决方法

首先锁定在同程网中的某个旅游页面，随后切换评论的页数，发现切换评论页面时发现是异步更新方式，如图 1.2 所示。此时浏览器出现 get 请求，我们利用 burpsuite 软件拦截这个请求的数据包，以便于分析它的 URL。如图 1.3 所示。

通过分析 URL，我们可以构造出能够返回 JSON 类型数据的 URL



图 1.2 切换页面

Figure1.2: Switch the paget

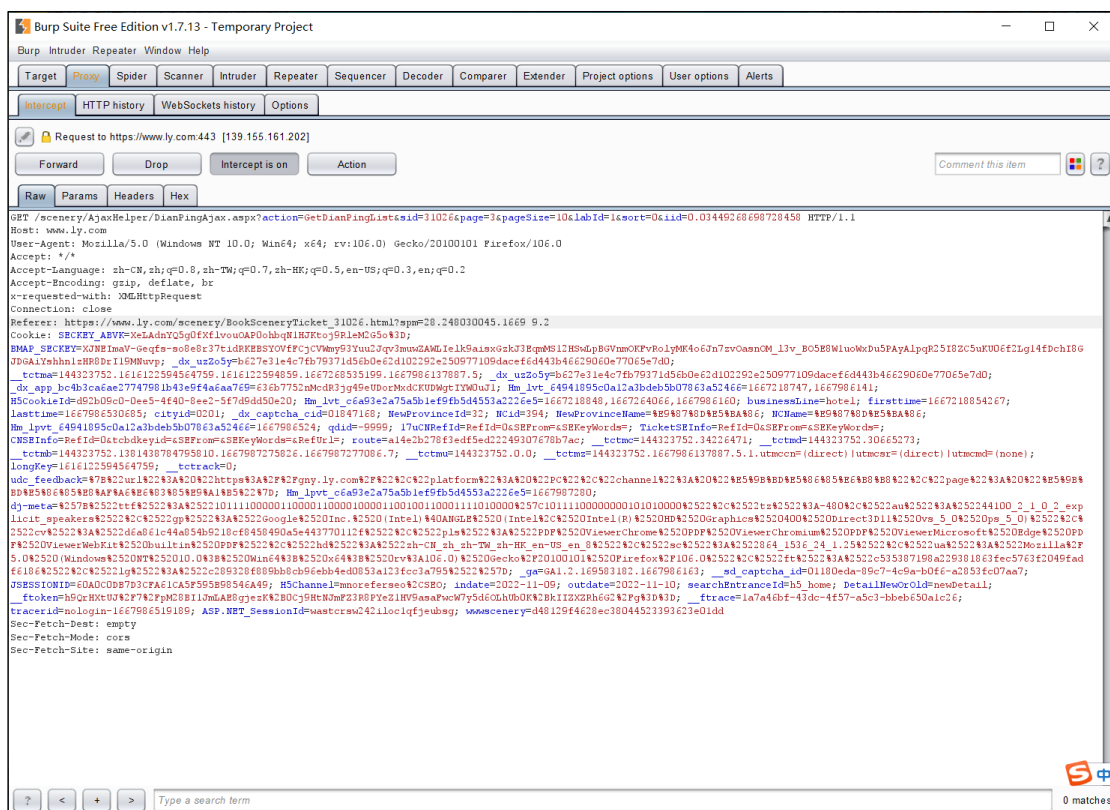


图 1.3 拦截数据包

Figure1.3: Block the packet

1.1.2 构造获取评论的 URL

① 分析获取评论的 URL

图 1.3 中 GET 请求的 URL 为：

/scenery/AjaxHelper/DianPingAjax.aspx?action=GetDianPingList&sid=31026&page=3&pageSize=10&labId=1&sort=0&iid=0.03449268698728458

我们甚至可以直接在浏览器中访问这个 URL，会发现它将返回一种 JSON 类型的数据（直接访问时要在 URL 最前面加上根域名“so.ly.com”）。在这个 JSON 数据中，我们可以得到我们需要的评论信息和评价信息。

这是一个传递了参数的 URL，sid 对应的数值代表了景点的 ID，page 对应

的数值代表了景区的页码。其余的参数我们保持不变即可。

所以，我们只要知道了景区的 ID，再依此遍历全部页码，就可以得到景区对应的所有评论了。

② 获取景区 ID

我们访问同程的主页，选择景点 → 国内景点 → 更多重庆景点，可以进入如下页面，如图 1.4 所示。

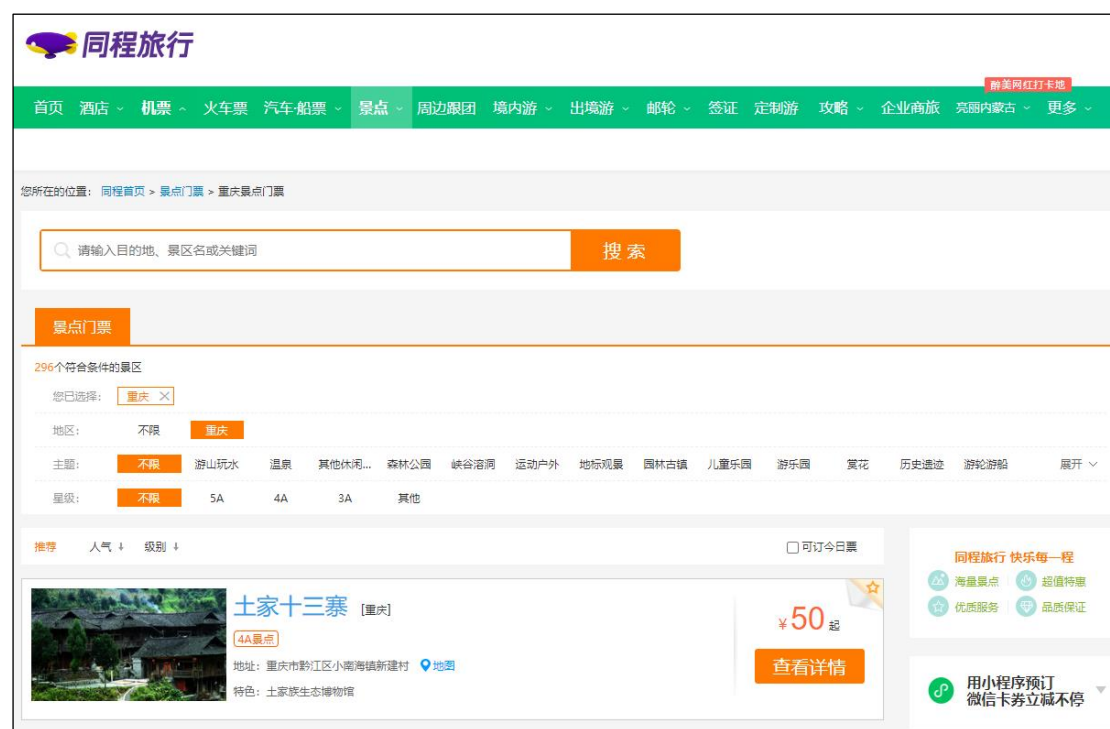


图 1.4 重庆全部景点页面

Figure 1.4: A page containing all the scenic spots in Chongqing

url: https://www.ly.com/scenery/scenerysearchlist_32_394__0_0_0_0_0_0_1001.html

经过反复切换页面，我们不难发现，URL 中的第二个 0 表示地区，最后的 1001 代表第 1 页，如 1002 代表第 2 页。

在知道网页的逻辑后，我们可以编写一个爬取所有经典编号的爬虫程序，见附件“分布式爬虫相关代码/test1/crawler.py”。

③ 构造最终的评论 URL

得到了景区的 ID 后，结合前面介绍的评论 URL 基本格式，我们就可以构造出完整的 URL 了，届时，我们会把这个 URL 存放到 Redis 数据库中。执行相关功能的程序见附件“分布式爬虫相关代码/test1/TaskDistribution.py”。

目的是为了随后我们直接从数据库中提取并请求这个 URL，便可以得到包含有评论和评价的 JSON 数据了。

数据库中的数据如图 1.5 所示。

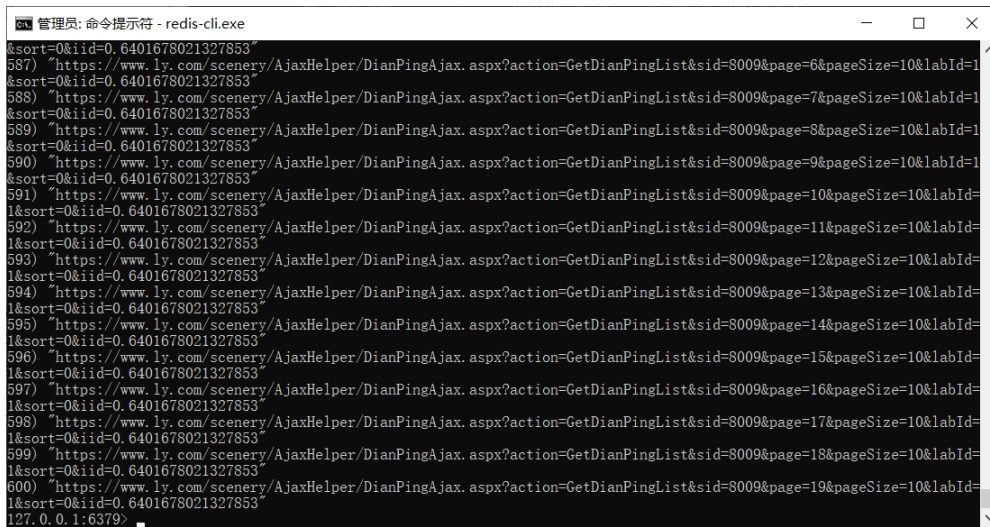


图 1.5 Redis 数据库中的 URL

Figure 1.5: URLs in the Redis database

1.1.3 本地环境设置与测试

① 初始化 redis

1) 在 Ubuntu 中安装 redis-server，执行如下指令即可。

```
sudo apt install redis-server
```

2) 关闭保护模式

进入 redis 数据库后，执行如下命令：

```
config set daemonize "no"
```

② 设置 Master 和 Slave

在 192.168.1.10 地址上，部署 scrapy-redis 作为 master（任务发布者）。在 192.168.1.11、192.168.1.12 上，部署 scrapy-redis 作为 slave（任务执行者）。

每一个 IP 地址下的终端都会有一个完全独立的 scrapy 爬虫项目，爬虫项目见附件“分布式爬虫相关代码”。区分 master 和 slave 的关键在于爬虫项目中，“分布式爬虫相关代码\test1\test1\setting.py”中的“REDIS_HOST”设置项。如图 1.6 所示。这里需要设置为主机的 IP 地址。

```
ITEM_PIPELINES = {
    'test1.pipelines.Test1Pipeline': 300,
    'scrapy_redis.pipelines.RedisPipeline': 400,
}

REDIS_HOST = '192.168.1.10'
REDIS_PROT = 6379
```

图 1.6 slave 中的 scrapy 项目的 setting

Figure 1.6: The Setting of scrapy project in slave

③ 爬取评论与评价

随后，编写一个爬虫程序，它的功能为读取 Redis 数据库中的 URL，根据 URL 获取 JSON 数据中的评论和评价，并将结果再保存回 Redis 数据库中。这个程序可以由 master 执行，也可以由 slave 执行，但是 slave 的本地不会保存这些数据，所有数据都会传给 master。

④ 主机查看数据

启动 slave 的爬虫项目后，我们在 master 端成功获取到了 slave 端传来的数据，如下图 1.7 所示。

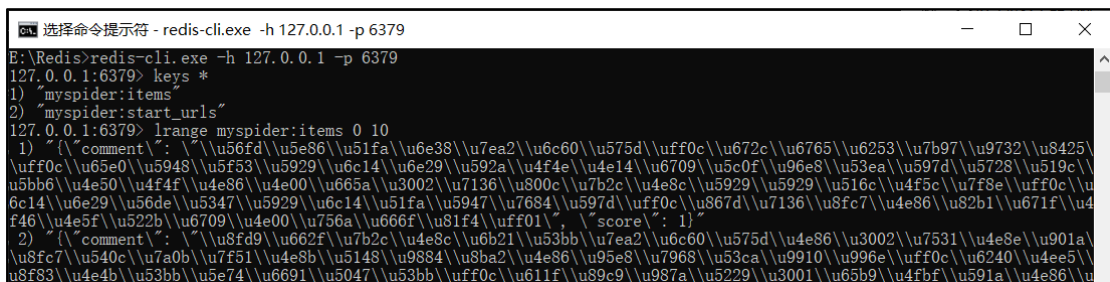


图 1.7 master 中查看 slave 爬取的数据

Figure 1.7: View data crawled by slave in master

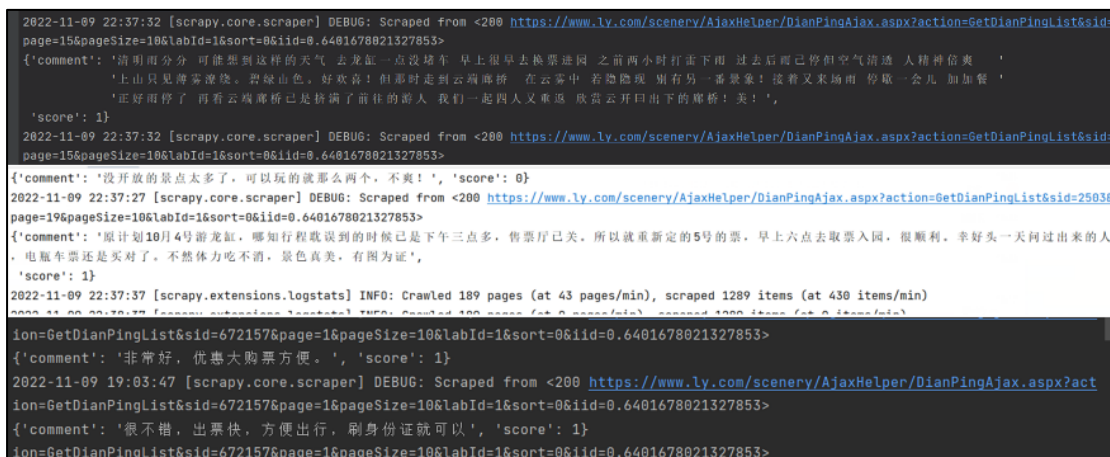


图 1.8 分布式爬虫的运行结果

Figure 1.8: The running results of the distributed crawler

1.1.4 多 PC 环境设置与测试

① 准备工作

基于 1.1.3 的环境部署思想，我们将三台 PC 机连入同一个局域网之下（比如某个手机热点）。选择 IP 地址为“192.168.25.231”的 PC 机作为 master，另外三台 PC 机作为 slave。

② 设置

每个 slave 都下载我们写好的爬虫项目，关闭所有防火墙，更改 setting 文件

中的“REDIS_HOST”的值为“192.168.25.231”。其余设置同 master 的设置一样。

③ 结果

最终我们的分布式爬虫成功在四台 PC 机上运行。执行结果如图 1.8 所示。

1.2 构造数据集

从图 1.7 中我们可以发现，此时我们得到的数据是一串未解码的数据，或者说是经过了序列化的数据。因此我们需要利用 python 读取并反序列化。

附件“RedisDataRead.py”的功能就是读取 redis 中的数据，并将其进行解码后用于构造数据集。并保存为 xls 格式的数据集。图 1.9 是我们收集的数据集的最终展示。

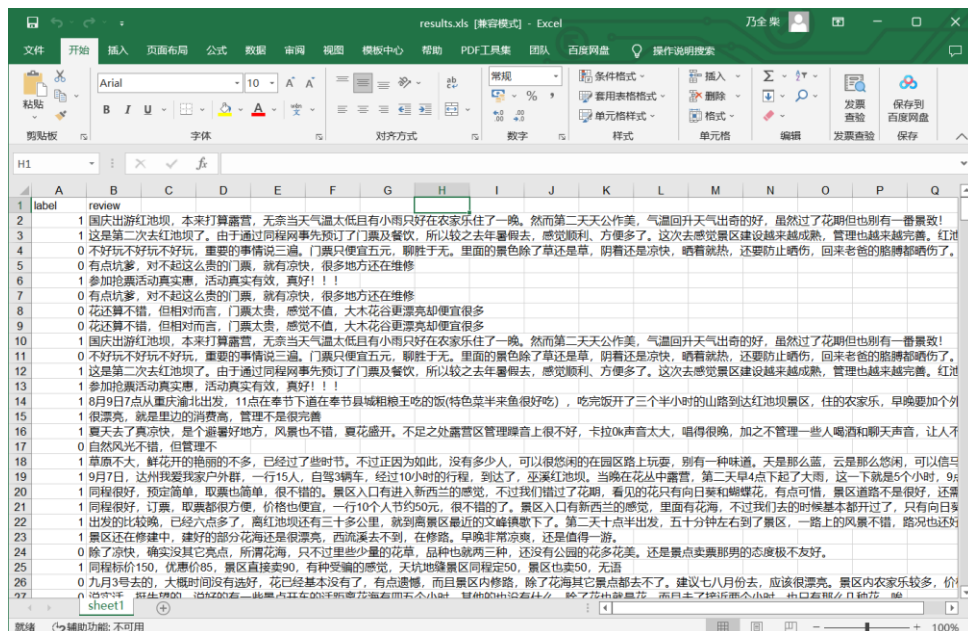


图 1.9 数据集展示

Figure 1.9: Data set presentation

1.3 数据集格式说明和使用方案

1.3.1 数据集格式说明

我们收集的数据集格式说明如下表 1.1 所示。

表 1.1 数据集格式说明

Table 1.1: Description of the data set format

数据集内容	数据规模	数据集格式	数据格式	预处理
同程网重庆全部景点评价	9 万	xls	情感态度 评价文本	需要

1.3.2 数据集使用方案

我们的数据格式仅有两个标签，即情感态度和评价文本。情感态度为 1 代表这是一个好评，情感态度为 0 代表这是一个差评。评价文本即游客的评价内容。

在使用数据集的时候，需要进行数据预处理工作。删除可能出现的重复数据；删除缺失了文本内容或者缺失了标签的数据；删除文本内容中的表情符号；限制最长文本内容。

1.4 小结

本章中，我们详细介绍了分布式爬虫项目的实现原理以及所采集数据库的格式说明和使用方案。现在对整个数据集收集的过程做出总结，其步骤如下表 1.2 所示。

表 1.2 数据集收集全过程汇总
Table 1.2: Description of the data set format

步骤序号	步骤内容
1	分析反爬手段
2	构造能够获取 JSON 类型数据评论的 URL，保存于 Redis 中
3	多 PC 设置分布式爬虫的环境
4	启动项目
5	构造数据集，反序列化
6	撰写数据集的格式说明与使用方案

需要注意，本章并没有给出数据集预处理的相关工作，我们在第三章的算法中会详细介绍。

2 环境搭建

在本章，我们将基于 Ubuntu18.04 系统搭建 Hadoop + Spark 环境。环境搭建完成后将运行一个简单的 word-count 算例用于验证我们搭建的环境是否成功。本章内容将会完整记录我们的平台搭建过程，并且会记录我们搭建过程中出现的错误，同时给出错误的解决方案。

2.1 环境搭建具体流程

2.1.1 安装 Ubuntu

本项目直接使用了微软应用商店中的 Ubuntu18.04.5 作为环境搭建所需的 Linux 系统。可以直接在微软应用商店下载。

2.1.2 安装 JDK

① JDK 的功能

JDK 是 Java 语言的软件开发工具包。JDK 是整个 JAVA 开发的核心，它包含了 JAVA 的运行环境和 JAVA 工具。我们的算法是基于 JAVA 实现，因此必须安装 JDK 才能保证整个项目的正常工作。

② 安装指令

```
sudo apt-get install default-jdk
```

注意，如果安装速度过慢的话需要考虑更换清华源。具体的更换方法不是本文的重点，就不再详细介绍。

③ 确定安装路径

确定安装路径的过程如下表 2.1 所示。

表 2.1 获取 JDK 安装路径的过程

Table 2.1 The procedure for obtaining the JDK installation path

命令行指令	我们的返回值
which java	/usr/bin/java
file /usr/bin/java	/etc/alternatives/java
file /etc/alternatives/java	/usr/lib/jvm/java-11-openjdk-amd64/bin/java

此时，我们就可以知道 JDK 的安装目录为 /usr/lib/jvm/java-11-openjdk-amd64。记住这个路径，一会儿会用到。

2.1.3 安装 hadoop

① 下载安装包

首先进入一个目录，我们在这个目录下载安装包。本次案例中的目录为用户根目录下的“bigData”文件夹。即“~/bigData”。

下载安装包的命令行指令为：

```
sudo wget https://mirrors.ustc.edu.cn/apache/hadoop/common/hadoop-3.2.3/hadoop-3.2.3.tar.gz
```

② 解压并配置环境变量.

1) 进入放有安装包的文件夹，执行下面这行代码可以解压刚才下载的安装包。我们定义文件解压后的所有内容放在文件夹“hadoop3”之下。

```
tar zxvf hadoop-3.2.3.tar.gz && mv hadoop-3.2.3 hadoop3
```

2) 配置环境变量

配置环境变量是为了更方便地访问 hadoop 中各个文件的路径。

```
echo "export HADOOP_HOME=/home/xiaodunshou/bigData/hadoop3" >> ~/.bashrc
echo "export PATH=/home/xiaodunshou/bigData/hadoop3/bin:$PATH" >> ~/.bashrc
source ~/.bashrc
```

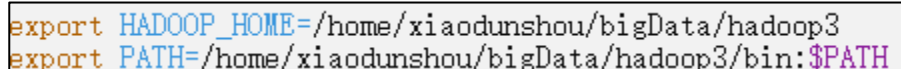


图 2.1 .环境变量配置成功后.bashrc 文件中应该存在的内容

Figure 2.1 What that should exist after environment variables are successfully configured

也可以在用户目录下输入 vim .bashrc 的指令直接进入与环境变量配置相关的文件进行修改。部分系统可能会将文件中的“\$PATH”自动替换为具体的路径，这可能导致我们的环境变量配置出错，无法正常使用。此时就必须进入“.bashrc”文件中手动配置环境变量了。配置成功后的“.bashrc”如图 2.1 所示，应该有图中的内容。

2.1.4 修改 hadoop 配置文件

① 修改配置文件：hadoop-env.sh

1) 进入文件所在目录

```
cd /home/xiaodunshou/bigData/hadoop3/etc/hadoop/
```

路径中的“xiaodunshou”是我们的虚拟机账户名。“bigData”是自建的文件夹，用于存放整个项目的相关文件。“hadoop3”是 hadoop 安装包解压后存放的文件夹。

2) 使用 vim 编辑器编辑文件

```
vim hadoop-env.sh
```

3) 搜索 JAVA_HOME 的对应项目，根据之前保存的 JDK 路径进行修改。

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

② 修改配置文件：core-site.xml

1) 使用 vim 编辑器编辑文件

```
vim $HADOOP_HOME/etc/hadoop/core-site.xml
```

2) 编辑内容

此时，我们在<configuration>中添加如下内容。我们会在每个<property>中用文字注释出这个标签的功能，但这不代表我们的代码中有相关的文字。后文的所有相关代码我们都会附上对应的文字注释。

```
<configuration>
```

```
<property>
```

设置 namenode 的 hdfs 协议的文件系统的通讯地址。

```
<name>fs.defaultFS</name>
```

```
<value>hdfs://localhost:9000</value>
```

```
</property>
```

```
<property>
```

设置代理权限，其他任何用户都可以通过 xiaodunshou 用户来访问我们的 hadoop 集群。

```
<name>hadoop.proxyuser.xiaodunshou.hosts</name>
```

```
<value>*</value>
```

```
</property>
```

```
<property>
```

设置代理权限，其他任何用户组都可以通过 xiaodunshou 用户来访问我们的 hadoop 集群。

```
<name>hadoop.proxyuser.wzt.groups</name>
```

```
<value>*</value>
```

```
</property>
```

```
</configuration>
```

③ 修改配置文件：hdfs-site.xml

1) 使用 vim 编辑器编辑文件

```
vim $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

2) 编辑内容

```
<configuration>
```

```
<property>
```

上传一个文件，并分割为 block 块后，每个 block 冗余副本数。

```
<name>dfs.replication</name>
```

```
<value>1</value>
```

```

    </property>
    <property>
        是否开启 hdfs 的 web 访问接口。
        <name>dfs.webhdfs.enabled</name>
        <value>true</value>
    </property>
    <property>
        是否启动 HDFS 文件的权限控制。
        <name>dfs.permissions.enabled</name>
        <value>false</value>
    </property>
</configuration>

```

④ 修改配置文件：yarn-site.xml

1) 使用 vim 编辑器编辑文件

```
vim $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

2) 编辑内容

```

<configuration>
    <property>
        设置结点管理器的通信方式。也可以说是在定义 Yarn 的执行机制。
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        设置集群在作业时产生的中间数据的存放路径。
        <name>yarn.nodemanager.local-dirs</name>
        <value>/tmp/yarn-local-dirs</value>
    </property>
    <property>
        结点管理器的主机 ip 地址。
        <name>yarn.nodemanager.hostname</name>
        <value>127.0.0.1</value>
    </property>
    <property>
        是否设置健康检查。我们为虚拟机环境，存储空间很小，如果设置健康检查可能导致中途分析中断。
        <name>yarn.nodemanager.disk-health-checker.enable</name>
        <value>false</value>
    </property>
    <property>
        Yarn 的管理员设置。如可以执行 yarn rmadmin/yarn kill 等命令时，该值必须配置，否则后续的队列相关的 ACL 管理员设置无法生效。
        <name>yarn.acl.enable</name>
        <value>0</value>
    </property>

```

```
</property>
</configuration>
```

2.1.5 安装 spark

① 下载安装包

首先进入一个目录，我们在这个目录下载安装包。本次案例中的目录为用户根目录下的“bigData”文件夹。即“~/bigData”。整个流程和 hadoop 的安装流程基本相同。

安装包的命令行指令为：

```
sudo wget https://archive.apache.org/dist/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
```

② 解压并配置环境变量

1) 进入放有安装包的文件夹，执行下面这行代码可以解压刚才下载的安装包。我们定义文件解压后的所有内容放在文件夹“spark3”之下。

```
tar zxvf spark-3.1.2-bin-hadoop3.2.tgz && mv spark-3.1.2-bin-hadoop3.2 spark3
```

2) 配置环境变量。在路径执行如下代码：

```
echo "export SPARK_HOME=/home/xiaodunshou/bigData/spark3" >> ~/.bashrc
```

执行完毕后，打开用户目录下.bashrc 文件，可以检查环境变量的修改结果，最底行的内容应该如图 2.2 所示。

```
export SPARK_HOME=/home/xiaodunshou/bigData/spark3
```

图 2.2 环境变量配置成功后.bashrc 文件中应该存在的内容

Figure 2.2 What that should exist after environment variables are successfully configured

2.1.6 修改 Spark 配置文件

① 修改配置文件：spark-env.sh

1) 进入相关文件夹

```
cd /home/xiaodunshou/bigData/spark3/conf/
```

2) 保存一个副本

```
cp spark-env.sh.template spark-env.sh
```

3) 用 vim 编辑器打开配置文件

```
vim spark-env.sh
```

4) 编辑内容

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/home/xiaodunshou/bigData/hadoop3
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_DIR=$HADOOP_HOME/etc/Hadoop
```

② 修改配置文件：yarn-site.xml

1) 进入相关文件夹

```
cd /home/xiaodunshou/bigData/hadoop3/etc/hadoop/
```

2) 使用 vim 编辑器打开配置文件

```
vim yarn-site.xml
```

3) 编辑内容：在原来<configuration>的基础上追加如下代码。

```
<property>
    每个容器请求分配的最大允许内存。
    <name>yarn.scheduler.maximum-allocation-mb</name>
    <value>1024</value>
</property>
<property>
    配置 mapreduce 的 shuffle 方法。
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
    是否启动物理内存量监控
    <name>yarn.nodemanager.pmem-check-enabled</name>
    <value>>false</value>
</property>
    是否启动虚拟内存量监控
<property>
    <name>yarn.nodemanager.vmem-check-enabled</name>
    <value>>false</value>
</property>
```

至此，Hadoop 的搭建圆满谢幕，。

2.2 测试用例具体流程

2.2.1 启动服务

① 启动 SSH 服务

1) 执行的命令行指令

```
sudo service ssh start
```

2) 可能的报错

当出现“permission denied”的错误时，依此执行如下三条命令。它的功能是实现 SSH 服务的免密码验证。

```
ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```



```
chmod 0600 ~/.ssh/authorized_keys
```

② 启动 Hadoop 集群

1) 执行的命令行指令

```
$HADOOP_HOME/sbin/start-all.sh
```

当显示如图 2.3 所示的内容时，代表 Hadoop 集群的启动成功了。

```
WARNING: Attempting to start all Apache Hadoop daemons as xiaodunshou in 10 seconds.  
WARNING: This is not a recommended production deployment configuration.  
WARNING: Use CTRL-C to abort.  
Starting namenodes on [MSI]  
Starting datanodes  
Starting secondary namenodes [MSI]  
Starting resourcemanager  
Starting nodemanagers
```

图 2.3 Hadoop 集群启动成功的控制台显示

Figure 2.3 A console showing that the Hadoop cluster has started successfully

2) 可能的报错

显示 localhost: datanode is running as process 10474. Stop it first and ensure /tmp/hadoop-xiaodunshou-datanode.pid file is empty before retry.

此时的错误原因为端口号已被占用，依此执行如下代码可以删除相关的占用信息：

```
cat /tmp/hadoop-xiaodunshou-* > kill  
> hadoop-xiaodunshou-datanode.pid  
> hadoop-xiaodunshou-namenode.pid  
> hadoop-xiaodunshou-nodemanager.pid  
> hadoop-xiaodunshou-resourcemanager.pid  
> hadoop-xiaodunshou-secondarynamenode.pid
```

2.2.2 上传数据集和准备程序

① 在 hdfs 中新建一个文件夹

1) 在 hdfs 中创建文件夹的命令行指令

需要注意这个文件夹不是创建在本地的虚拟机环境中的，而是创建在 hdfs 专门的云端中的。

```
hdfs dfs -mkdir -p input
```

创建完成后，可以通过如下指令查看是否创建成功了。

```
hdfs dfs -ls
```

2) 上传数据集

假设数据集放在了用户目录下。此时执行如下命令行指令即可。

```
hdfs dfs -put ~/ sample.txt input/
```

可以通过如下命令查看已经上传的数据集。

```
hdfs dfs -cat input/sample.txt
```

② 将 jar 程序移动到本地

我们将用 JAVA 写好的 jar 程序放在了 windows 系统中的桌面上，然后现在需要将这个 jar 程序转移到虚拟机中，执行如下代码即可。这里的代码不唯一，根据我们的桌面安装路径有变化。

```
cd /mnt/d/0__Users/Admin/Desktop/DeskTop
```

```
mv WordCount.jar ~/bigData
```

此时，WordCount.jar 程序就成功复制到了 ubuntu 系统中用户目录下的 bigData 项目文件夹中。随后，我们进入项目文件夹。

2.2.3 执行 WordCount 算例

① 执行 WorkCount 程序

```
hadoop jar WordCount.jar input/sample.txt ./result
```

在这条命令中，WordCount.jar 是我们的程序，input/sample.txt 是数据集存放的路径，./result 是结果保存的路径。

② 查看结果

1) 查看 result 文件夹中是否有我们的结果

```
hdfs dfs -ls result
```

此时能够得到如图 2.4 所示的结果。

```
xiaodunshou@MSI:~/bigData$ hdfs dfs -ls result
Found 2 items
-rw-r--r--  1 xiaodunshou supergroup          0 2022-11-16 11:37 result/_SUCCESS
-rw-r--r--  1 xiaodunshou supergroup    8729 2022-11-16 11:37 result/part-r-00000
```

图 2.4 查看是否生成了统计结果文件

Figure2.4 Check whether the statistics file is generated

可以看到 result 文件夹下此时有两个文件，第一个文件为自动生成的空文件，目的是展示程序执行的状态，状态为 SUCCESS，可见统计成功。第二个文件为最终的统计结果。

2) 查看结果

```
hdfs dfs -cat result/part-r-00000
```

我们的统计结果如下图 2.5 所示.

```
xiadunshou@MSI:~/bigData$ hdfs dfs -cat result/part-r-00000
      14
" calculating 1
" 1945 1
" I 1
" agent 1
" complete 1
" father 1
" full 1
" modern 1
" programmed 1
" one 1
" programmable 1
(CPU) 1
(IC) 1
(MOS 1
(as 1
(automaton) 1
(clay 1
(computation) 1
(control 1
(e.g., 1
(keyboards, 1
(later 1
(main 1
(monitor 1
```

图 2.5 word-count 运行结果
Figure 2.5 word-count computational results

至此，我们可以得出最终结论。Hadoop + Spark 的环境搭建成功，word-count 算例运行成功。

2.3 小结

本章中，我们在 Windows 自带的 Ubuntu 系统搭建了一个基于 Hadoop + Spark 大数据框架。并通过一个 Wordcount 算例验证了我们的环境正确无误。现在对整个环境的搭建作出总结，见表 2.2。对算例验证的实现流程作出总结，见表 2.3。

表 2.2 Hadoop 和 Spark 环境搭建流程汇总
Table 2.2 Hadoop and Spark environment to build process summary

步骤	步骤内容	控制台指令
1	安装 Ubuntu	/
2	安装 JDK	sudo apt-get install default-jdk
3	安装 Hadoop	sudo wget https://mirrors.ustc.edu.cn/apache/hadoop/common/hadoop-3.2.3/ hadoop-3.2.3.tar.gz
4	解压安装包并配置环境变量	/
5	修改 Hadoop 配置文件	vim \$HADOOP_HOME/etc/hadoop/hadoop-env.sh
		vim \$HADOOP_HOME/etc/hadoop/core-site.xml
		vim \$HADOOP_HOME/etc/hadoop/hdfs-site.xml
		vim \$HADOOP_HOME/etc/hadoop/yarn-site.xml

续表 2.2

步骤	步骤内容	控制台指令
6	安装 Spark	<code>sudo wget https://archive.apache.org/dist/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz</code>
7	解压安装包并配置环境变量	<code>/</code>
8	修改 Spark 配置文件	<code>vim \$SPARK_HOME/conf/spark-env.sh</code> <code>vim \$HADOOP_HOME/etc/hadoop/yarn-site.xml</code>

表 2.3 Wordcount 执行流程汇总

Table 2.3 Wordcount performs the process summary

步骤	步骤内容	控制台指令
1	启动 SSH 服务	<code>sudo service ssh start</code>
2	启动 Hadoop 集群	<code>\$HADOOP_HOME/sbin/start-all.sh</code>
3	上传数据集	<code>hdfs dfs -mkdir -p input</code> <code>hdfs dfs -put ~/ sample.txt input/</code>
4	上传程序	<code>cd /mnt/d/0__Users/Admin/Desktop/DeskTop</code> <code>mv WordCount.jar ~/bigData</code>
5	执行程序	<code>hadoop jar WordCount.jar input/sample.txt ./result</code>
6	查看结果文件	<code>hdfs dfs -ls result</code>
7	查看结果内容	<code>hdfs dfs -cat result/part-r-00000</code>

3 算法

本章我们将介绍并行化算法的设计以及算法执行的结果。整个算法将以 Hadoop 的分布式管理系统和资源管理器为框架，以 Spark 的 pySpark 工具作为算法执行和调试的基础。我们使用支持向量机 SVM 算法来实现我们的最终功能。算法执行成功后，我们会尝试该算法的分布式执行。

3.1 准备工作

① 启动服务

启动 SSH 服务和 Hadoop 集群，它的控制台代码前文已经介绍过：

```
sudo service ssh start
```

```
$HADOOP_HOME/sbin/start-all.sh
```

② 准备程序与数据集

```
mkdir ~/bigData/algorithm_application
```

```
mv /mnt/d/0__Users/Admin/Desktop/DeskTop/dataset.csv ~/bigData/algorithm_application
```

```
mv /mnt/d/0__Users/Admin/Desktop/DeskTop/app.ipynb ~/bigData/algorithm_application
```

③ 启动可视化界面

要执行整个算法，我们必须为我们的 Ubuntu 系统安装图形化界面。这个部分不是本文的重点，不再详细介绍，相关内容可见附件“03 其他可参考资料/03 基于 WSL 的 Ubuntu 系统安装(非应用商店直接下载).pdf”

安装后图形界面后，执行如下控制台命令启动图形界面服务：

```
startxfce4
```

随后会自动启动 Ubuntu 的可视化模块。

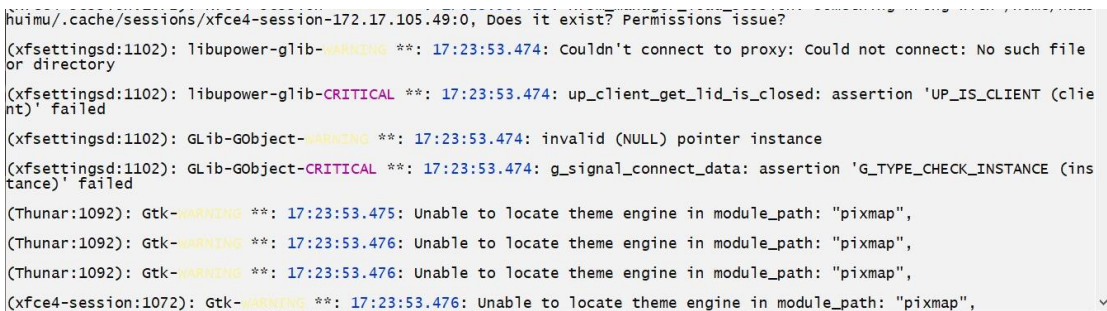


图 3.1 pySpark 的启动

Figure 3.1.PySpark launching

④ 启动 pySpark 工具

直接执行如下代码即可。启动后的控制台显示情况如图 3.1 所示。

```
pyspark
```

3.2 算法原理

pySpark 的控制台指令执行后，我们会自动跳转到一个 Jupyter notebook 的界面中去。可以在这个界面中实现我们的所有的算法原理以及可视化的展示。

3.2.1 加载数据集

导入相关工具包后的第一步，自然是要加载数据集了。注意这个代码执行之前，必须保证我们的数据集文件已经成功上传到了 HDFS 文件管理系统中去。

代码如图 3.2 所示。

```
dataset_1 = spark.read \
    .format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load('input/a1.csv')

dataset_2 = spark.read \
    .format("csv") \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .load('input/a2.csv')

#数据转为dataframe并设置字段值
dataset_1 = dataset_1.toPandas()
dataset_1.columns = ['id', 'target', 'text']
dataset_1.drop(['id'], axis=1)

#数据转为dataframe并设置字段值
dataset_2 = dataset_2.toPandas()
dataset_2.columns = ['id', 'target', 'text']
dataset_2.drop(['id'], axis=1)

dataset = pd.merge(dataset_1, dataset_2, how="outer")
dataset_p = dataset
print(dataset)
```

图 3.2 加载数据集

Figure 3.2: Load dataset

3.2.2 数据预处理

数据预处理中，我们需要执行去除缺失值、去除留言重复值、索引重排、汉字分词、去除停用词、更新数据框的工作。代码如图 3.3 所示。

对于存在缺失值的项目，我们直接将其删除。由于我们的算法是回归模型，因此一定不能有重复值的存在。汉字分词，我们使用了“jieba”工具。索引重排是因为我们删除了部分数据，索引就不再连续了。更新数据框是指，我们在使用分词工具后，分词结果是在原数据集的基础上多加了一列，因此必须对原数据集的那一列进行删除。

数据预处理的相关代码见图 3.3。

```

# 数据去掉缺失值
dataset_z = dataset_p.dropna()

# 数据去掉留言重复值
dataset_z.drop_duplicates(keep='first', inplace=True, subset='text')

# 索引重排
dataset_z = dataset_z.reset_index()

# 文本分词, 去除停用词
text = dataset_z.text.values.tolist()
words = [" ".join(jieba.lcut(i)) for i in text]
new_text = pd.core.frame.DataFrame({"text":words})

# 更新数据框
dataset_z.drop(["text"], axis=1)
dataset_z["text"] = new_text

```

图 3.3 数据预处理

Figure 3.3: data preprocessing

数据预处理结束后, 我们需要将数据上传到“Spark sql”中去。这是 Spark 处理结构化数据专门的模块。相关的代码见图 3.4。

```
dataset_df = sqlContext.createDataFrame(dataset_z)
```

图 3.4 传入 Spark sql

Figure 3.4 Upload to Spark sql

3.2.3 特征提取管道

PySpark 的算法是基于“管道 (pipeline)”这一结构进行的。一个管道有若干个阶段组成, 这个阶段只能是转换器 (Transformer) 或评估器 (Estimator)。转换器会将输入的数据集 (DataFrame) 转换为带有预测标签的另一种数据集。这会伴随着一个新的列输出。而评估器可以看作是一个初始化, 每一次执行转换器操作时, 都必须提前执行一个评估器的声明操作, 即建立模型。

```

#文本分词到words字段
tokenizer = Tokenizer(inputCol="text", outputCol="words")

#对words字段计算词频并transform到tf字段
hashtf = HashingTF(numFeatures=256, inputCol="words", outputCol='tf')

#对tf字段计算tf-idf, 降低多文档高频词权重
idf = IDF(inputCol='tf', outputCol="rawfeatures", minDocFreq=5)

#将上述操作放到pipeline中以便使用
pipeline = Pipeline(stages=[tokenizer, hashtf, idf])

```

图 3.5 定义特征提取管道

Figure 3.5 Define the feature extraction pipeline

PySpark 并行化计算的基础就是管道操作。如图 3.5 所示。

定义特征提取管道后，对管道执行 fit 和 transform 运算，就能得出最终的结果。管道的运算原则是以列优先，根据管道中的“stages”的定义依此执行三个 PySpark 内置的方法“tokenizer”、“hashtf”、“idf”。“tokenizer”的功能是对原句子按照空格进行分词，它的输入是原始数据集。“hashtf”的功能是对词频进行统计，即计算 tf 参数，它的输入是“tokenizer”的输出。“idf”的功能是计算 tf-idf 值，并降低多文档高频词的权重，它的输入是“hashtf”的输出。

提取的具体执行如下 3.6 图所示。

```
# 特征提取
pipelineFit = pipeline.fit(dataset_df)
dataset_df = pipelineFit.transform(dataset_df)

dataset_df.show(5)
```

图 3.6 特征提取

Figure 3.6 Feature extraction

可见，先执行了 fit 方法，随后再执行了 transform 方法。经过实验证明，这个顺序不可对调，不可减少任何一个步骤。

这个代码逻辑可以这样理解（不一定准确）：首先要定义一个管道对象。通过对 Pipeline 类对象调用 fit 方法生成这样的管道对象，Pipeline 类对象在建立时，就需要声明了这个管道中有哪些方法；随后，对管道对象调用 transform 方法，就可以让输入的数据集以此执行管道中的所有方法，最终产生输出。

本案例最终的输出结果如图 3.7 所示。可见，经过了“tokenizer”方法，原数据集产生了一个“word”字段；经过了“hashtf”方法，原数据集产生了一个“tf”字段；经过了“idf”方法，原数据集产生了一个“rawfeature”字段。

index	target	text	words	tf	rawfeatures
0	1	专门 选在 工作日 过来 体验 一...	[专门, 选在, 工作日, 过来, ...]	(256, [2, 10, 20, 21, ...])	(256, [2, 10, 20, 21, ...])
1	1	这次 是 夜场 换票 的 人 很多...	[这次, 是, 夜场, 换票, 的, ...]	(256, [4, 5, 6, 8, 9, 1, ...])	(256, [4, 5, 6, 8, 9, 1, ...])
2	1	评价 有点 晚了, 完全 是 ...	[评价, 有点, 晚, 了, , , ...]	(256, [1, 2, 16, 18, 2, ...])	(256, [1, 2, 16, 18, 2, ...])
3	1	一块钱 能玩到 已经 很 不错 了...	[一块钱, 能玩到, 已经, 很, ...]	(256, [1, 4, 7, 11, 16, ...])	(256, [1, 4, 7, 11, 16, ...])
4	1	距 主城 温泉 水质 最棒 哒 哒...	[距, 主城, 温泉, 水质, 最, ...]	(256, [7, 23, 32, 34, ...])	(256, [7, 23, 32, 34, ...])

only showing top 5 rows

图 3.7 特征提取后的数据集

Figure 3.7 The dataset after feature extraction

3.2.4 建立 SVM 模型

① 数据的最后准备

在这个步骤中，我们需要设定数据标签，对特征进行标准化，处理掉那些无用的字段，对数据集进行分割。代码如图 3.8 所示。


```
# 设定数据标签
labelIndexer = StringIndexer(inputCol="target", outputCol="label").fit(dataset_df)
dataset_df = labelIndexer.transform(dataset_df)

# 特征标准化
normalizer = StandardScaler(inputCol="rawfeatures", outputCol="features")
normalizer_model = normalizer.fit(dataset_df)
dataset_labeled_df = normalizer_model.transform(dataset_df)

# 去除前面处理遗留的无用字段
dataset_labeled_df = dataset_labeled_df.select("features", "label")

# 数据集分割
(training_data, test_data) = dataset_labeled_df.randomSplit([0.7, 0.3])
```

图 3.8 建模前的数据准备

Figure3.8 Data preparation before modeling

从代码中可以看到，前两个小步骤中，和管道类似，它同样是使用了 `fit` 和 `transform` 两个方法。因为 PySpark 的工作是完全基于管道来创造工作流的，所以哪怕是设定数据标签这种及其简单的工作它也要经过管道来进行。这里我们没有定义专门的 Pipeline 类，通过 `stage` 参数来放置两个方法。但是我们认为“StringIndexer”或“StandardScaler”这种 PySpark 内置的算法工具类，它从本质上来看，就是一个 `stage` 为 1 的管道，底层代码中应该是继承了相关的父类的。

② 建模并训练、预测

同理，我们也会在这里定义一个放置了 SVM 算法的，`stage` 为 1 的管道，并对它执行 `fit` 和 `transform` 两个方法。这一部分的代码如图 3.9 所示。

```
# 配置模型参数
dtree = lr = LinearSVC(maxIter=256, threshold=0.5)

# 模型训练
model = lr.fit(training_data)

# 模型检验
predictions = model.transform(test_data)
```

图 3.9 建模并训练、预测

Figure3.9 Model and train and predict

3.2.5 模型评估

我们对于模型的评估值只要有两个指标。即准确率和 F1 值。准确率是指预测结果属于某一类的个体，实际属于该类的比例。F1 值是准确率和召回率的调和平均数，召回率是指被正确预测为某个类别的个体数量与数据集中该类别个

体总量的比例。

模型评估使用了一个单独的模块，相关代码的结果如下图 3.10 所示。

```
#导入评估模块
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

#计算准确率
acc_evaluator_dt = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
accuracy_dt = acc_evaluator_dt.evaluate(predictions)
print("Acc: %g" % (accuracy_dt))

#计算F1值
f1_evaluator_dt = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
f1_score_dt = f1_evaluator_dt.evaluate(predictions)
print("F1: %g" % (f1_score_dt))
```

图 3.10 模型评估

Figure3.10 Model evaluation

3.3 结果分析与可视化

我们最终的模型评估结果，准确率为 85.64%，F1 为 79.01%。情感分类作为一个经典的分类问题，这样的评估结果可以说是不够理想的。我们猜测是我们选择的机器学习算法 SVM 在这里的适用效果可能不够好。因此，我们需要对原数据的特征进行 PCA 降维后再进行可视化的展示，来验证我们的猜测。

可视化的代码如 3.11 所示。

```
#将标准化特征字段数据的SparkVector类型转为Array类型
from pyspark.sql.functions import UserDefinedFunction
def convert_to_dense(col):
    return DenseVector(col.toArray())
convert_to_dense_udf = UserDefinedFunction(lambda x: convert_to_dense(x), VectorUDT())
test_dataset_labeled_df = test_data.withColumn("dense_features", convert_to_dense_udf(col("features")))

#利用PCA对Array类型标准化特征进行降维
PCA_m = PCA(k=2, inputCol="dense_features", outputCol="pcaFeatures")
PCA_model_cluster = PCA_m.fit(test_dataset_labeled_df)
test_data_points = PCA_model_cluster.transform(test_dataset_labeled_df).select("pcaFeatures", "label")

#获取绘图数据
test_data_points_l = test_data_points.rdd.map(lambda x: (x[0].toArray(), x[1])).collect()
test_x = list(map(lambda x: x[0][0], test_data_points_l))
test_y = list(map(lambda x: x[0][1], test_data_points_l))
test_class = list(map(lambda x: x[1], test_data_points_l))
colors = ['red' if label == 1 else 'blue' for label in test_class]

#去除奇异值
def del_point(nums=1):
    for _ in range(nums):
        del_index = test_x.index(max(test_x))
        del test_x[del_index]
        del test_y[del_index]
        del colors[del_index]
        del_index = test_x.index(min(test_x))
        del test_x[del_index]
        del test_y[del_index]
        del colors[del_index]
        del_index = test_y.index(max(test_y))
        del test_x[del_index]
        del test_y[del_index]
        del colors[del_index]
        del_index = test_y.index(min(test_y))
        del test_x[del_index]
        del test_y[del_index]
        del colors[del_index]
del_point(10)

#绘图
plt.scatter(test_x, test_y, c = colors)
```

图 3.11 可视化代码

Figure3.11 Visualization code

可视化执行完成后的结果如下图 3.12 所示。

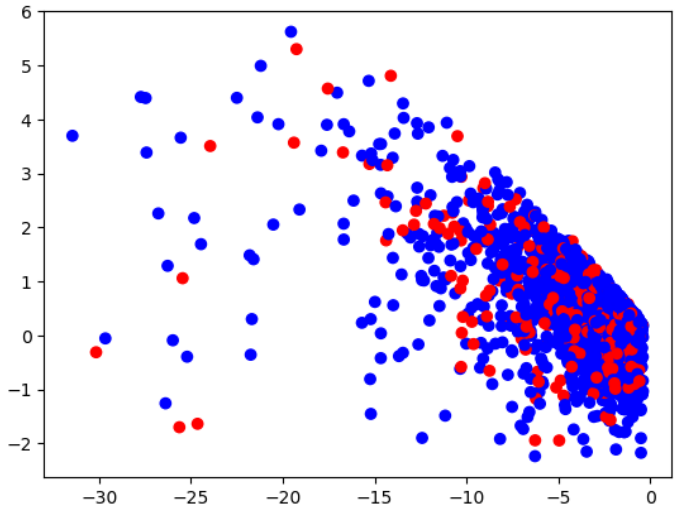


图 3.12 特征降维后的可视化展示

Figure3.12 Visual representation of features after dimensionality reduction

PCA 降维特征为 2 维后，绘制在图中。红色点代表差评，蓝色点代表好评。可以发现差评均匀地分布在了整个空间中，因此要想使用单纯的线性回归类型的算法，去找到一条直线，或者说是找到一个超平面去对两种类型的数据进行一个完美的划分，是比较困难的。

对于情感分类而言，使用深度学习算法能够取得更好的成果，但是要想在 PySpark 中应用深度学习算法是比较困难的，虽然可以直接调用相关的工具包，但是这样就无法实现大数据思维中的并行化加算了。因为只有基于 PySpark 的管道结构建立的算法，才能真正实现并行化计算的逻辑。

3.4 小结

本章中，我们基于我们搭建的环境，将我们爬取的数据集交给了 pySpark 进行运算，使用的算法为 SVM。现在对整个算法的流程作出总结，见表 3.1。

表 3.1 算法流程

Table3.1 Algorithm flow

步骤	步骤内容
1	启动相关服务与可视化界面
2	加载数据集
3	数据集预处理：去除缺失值、去除留言重复值、汉字分词、去除停用词等
4	定义特征提取管道：“tokenizer”、“hashtf”、“idf”
5	执行特征提取算法：fit(); transform()

续表 3.1

步骤	步骤内容
6	模型前数据预处理：标签指定、特征预处理、删除无用数据、数据集划分
7	建立 SVM 模型：LinearSVC()
8	训练模型
9	模型评估：准确率、F1
10	可视化分析：PCA 降维

4 创新性思路和策略

我们在整个项目中，认为创新点有两点。

① 爬虫的分布式运行

我们使用 **Scrapy-Redis** 框架成功建立起了分布式爬虫。在多台 PC 机上共同爬取数据。经过测试，在分布式爬虫框架下，爬取速率随 PC 机数目的增多线性增长。如下图 4.1 所示。

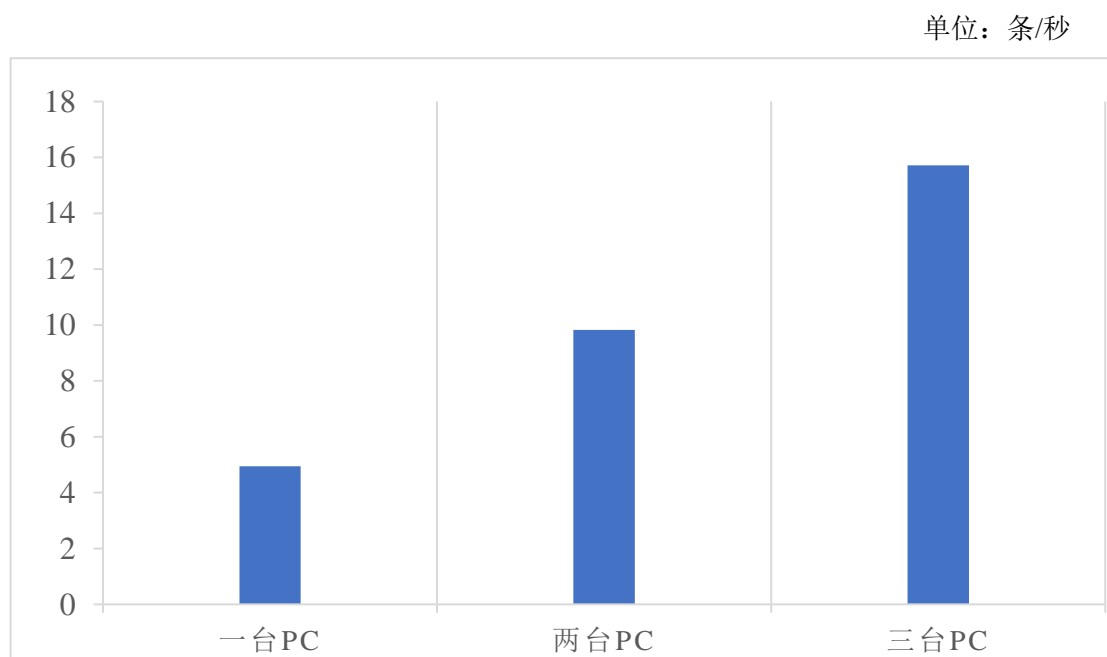


图 4.1 分布式爬虫带来的效率提升

Figure 4.1 Efficiency improvement of distributed spider

② PySpark 的使用

在算法的实现中，我们选择了 **PySpark** 作为我们算法开发的 IDE。基于 **PySpark** 的开发是并行化计算的前提条件。而在最后一次研讨课中，我们认为其他小组基于 **Pycharm** 的算法开发虽然能支持更高级更复杂的算法，但是他们在并行化计算的实现中，一定会遭遇困难。

尽管由于时间紧迫，科研任务繁重，我们没能给出并行化计算的成功案例，但是我们经过项目前期的方法调研和理论分析，能得出一个结论，即我们基于 **PySpark** 的算法实现，在后续的优化之后，是一定可以满足并行化计算这个大数据课程的核心思想的。

5 总结与展望

5.1 项目带来的收获

整个大数据项目中，我们学习了分布式爬虫的使用、大数据框架的部署、大数据算法的应用。这对于我们开拓视野、提高实践能力，都有着至关重要的作用。这对于我们未来想要从事大数据行业的同学来说，是一系列非常有意义的经历。

我们发现，很多大厂的招聘需求中，都对大数据框架的掌握提出了一定的需求。假如我们未来想要从事相关的岗位，而本项目又让我们对大数据常见框架的部署和使用有了一定的了解，这毫无疑问会增强我们的竞争力，在当今这个大内卷时代带来优势。

5.2 课程带来的收获

在课程的学习中，首先学习了大数据的一些宏观的理论知识。比如大数据的 4V 特征：体量大、种类多、速度快、价值密度低。比如据的技术支撑为计算、存储、智能。再比如的应用渗透到了各行各业。基于这些宏观的知识，我们脑海中便形成了大数据的基本框架。

学习了 Hadoop 这一经典大数据框架的相关概念。知道了它的两个核心组成成为“分布式文件系统（HDFS）”和“分布式计算系统（MapReduce）”。我们知道了 HDFS 中主节点和从节点的工作原理，知道了冗余数据的保存方法，知道了资源调度其 YARN 的实现机理，还知道了 MapReduce 算法的实现方法。这便是我们和那些普通开发人员的区别，我们真正做到了“知其所以然”。

有了前面的铺垫，我们很轻松地入门了分布式数据库（Hbase）和 Spark 这一新的大数据系统。我们知道了 Hbase 因其强大的扩展能力，是能够应对新时代数据爆炸增长的数据库。而 Spark，则能给我们带来更加灵活的编程模型，能在各种应用场景下使用：如批处理、交互式查询、流数据处理等。

最后，我们了解到了一些常见的大数据算法和大数据应用。至此，我们对大数据这门课的整个思维框架就得到了完美的建立。感谢老师给我们带来了这样一门精彩的课程。

附录 A：小组分工

我们的小组分工如下表 A.1 所示。

表 A.1 小组分工

Table A.1 Group division

工作内容				
平台搭建			✓	✓
测试用例		✓		
分布式爬虫	✓		✓	
数据预处理	✓			✓
并行算法编写	✓			
可视化	✓	✓		
文档撰写		✓		
PPT 制作			✓	✓
研讨课讲解 1			✓	
研讨课讲解 2	✓			
研讨课讲解 3		✓		
资料整理	✓	✓		