**高豪-实验三报告**

电子科技大学信息与软件工程学院

**项 目 报 告**

课程名称 大数据分析与智能计算

理论教师 黄俊

实验教师 黄俊

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**项 目 报 告**

**姓名：**高豪  **指导教师：** 黄俊  **地点：**第二教学楼106

1. **项目名称：**航空公司延误和取消分析项目
2. **项目时间：**2022.12.2—2022.1.7
3. **项目原理**

spark是一个用来实现快速，通用的集群计算平台。spark能够在内存中进行计算，比需要多次磁盘读写的MapReduce快；spark可以进行批处理、迭代算法、交互式查询、流处理等；spark不仅可以提供基于python、java、Scala和SQL的API，还能和其他大数据工具密切配合使用，spark可以运行在hadoop集群上。

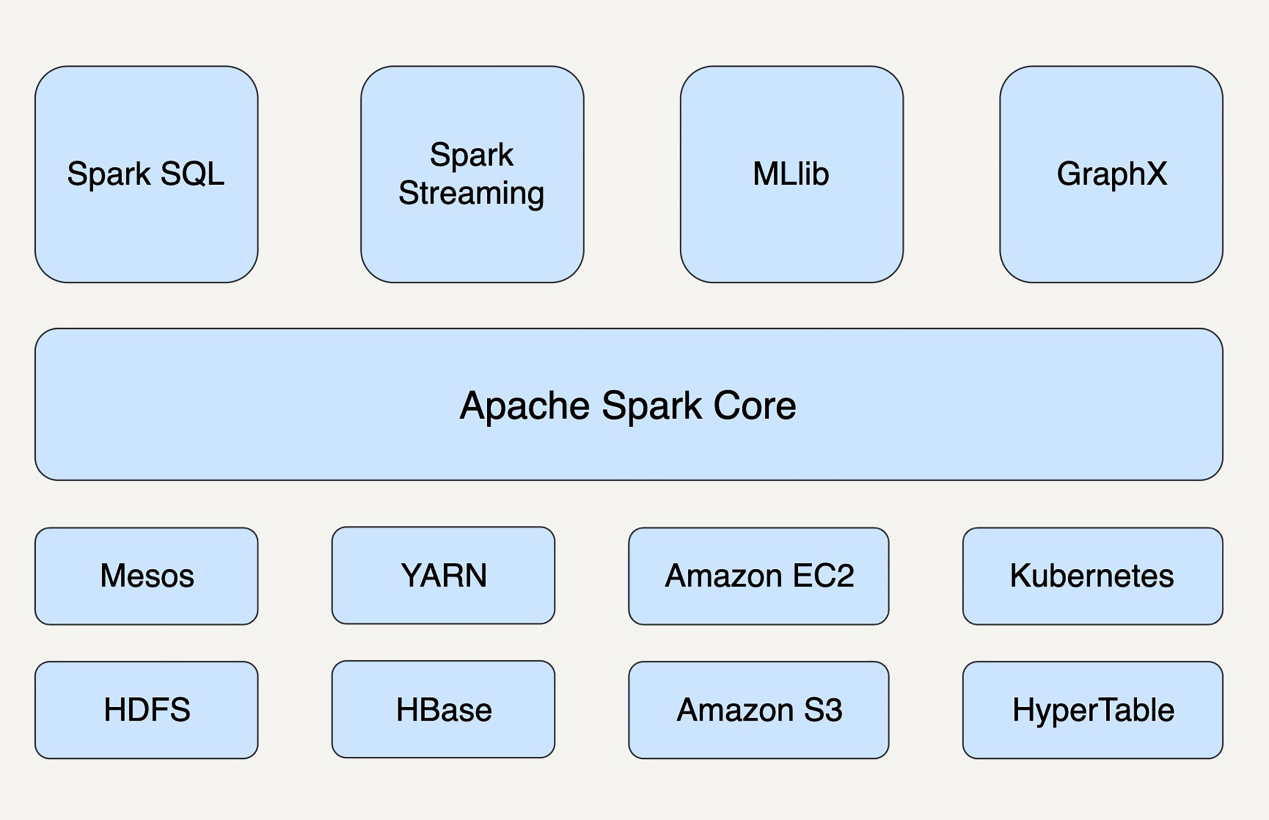


图1 Apache Spark应用架构图

Core库中主要包括上下文（Spark Context）、抽象数据集（RDD）、调度器（Scheduler）、洗牌（shuffle）和序列化器（Serializer）等。Spark系统中的计算、IO、调度和shuffle等系统基本功能都在其中。在Core库之上就根据业务需求分为用于交互式查询的SQL、实时流处理Streaming、机器学习Mllib和图计算GraphX四大框架，除此外还有一些其他实验性项目如Tachyon、BlinkDB和Tungsten等。这些项目共同组成Spark体系结构，当然Hadoop中的存储系统HDFS迄今仍是不可被替代，一直被各分布式系统所使用，它也是Spark主要应用的持久化存储系统。

PySpark是Spark提供的Python接口，利用PySpark，调用其中的接口，可以实现航班延误数据集的数据预处理、数据分析与数据建模预测。通过结合Jupyter Notebook，可以在Web界面输入Python命令后立刻看到结果，还可将数据分析的过程和运行后的命令与结果存储成笔记本。

1. **项目内容**
2. 本机安装并测试PySpark（在前序实验中要求已完成hadoop伪分布式配置以及spark的安装）；
3. 安装并测试Jupyter Notebook；
4. 启动运行PySpark对航班延误数据集进行分析计算。
5. **需求分析与设计**

项目功能性需求：

1. 查看飞机延误时间最长的前10名航班；
2. 计算延误的和没有延误的航空公司的比例；
3. 分析一天中、一周中延误最严重的飞行时间；
4. 分析短途航班和长途航班，哪种航班取消更严重；
5. 建立机器学习算法模型，预测未来航班取消情况。

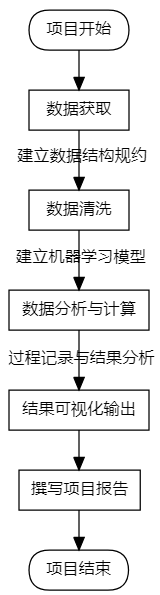
项目要求使用python编程语言以及使用Pyspark计算框架完成航班延误及取消的数据分析以及计算任务。要求精确的实现以上5个功能任务点，并使用图、表对结果进行可视化显示（使用Jupyter Notebook可以通过Web页面动态的显示程序运行结果）。

项目非功能性需求：

1. 对于功能任务点1-4，需要在2秒内完成数据计算与统计；
2. 对于功能任务点5，机器学习算法模型识别准确率要求在95%以上。

项目设计：

以下为项目工程开展流程图：



项目工程开展流程图

1. **项目计划**

项目开始后，首先进行数据挖掘与抓取，获得航班延误数据集；然后建立数据结构规约（具体规约见“项目结果与分析”）进行数据清晰得到“干净的”、有序的、符合数据分析计算的数据集；接下来建立机器学习模型/深度学习神经网络（本小组使用LSTM长短期记忆神经网络）模型，进行训练分析；最后对结果进行可视化展示并完成项目过程记录与结果分析报告。

1. **项目环境配置管理**
2. **操作系统**： ubuntu-20.0.4、windows10
3. **开发工具**：Pyspark，Jupyter Notebook，Pycharm
4. **配置过程**

首先确认Spark的环境是否完成伪分布式配置（实验一要求的spark环境），输入以下命令，登录hadoop用户并查看spark环境变量配置是否正确：

**$ su – hadoop**

（输入hadoop用户登录密码，该密码将不会以明文回显至屏幕）

**$ echo $SPARK \_HOME**

hadoop用户登录成功，spark环境变量显示正确。环境配置文件具体内容如图2所示：

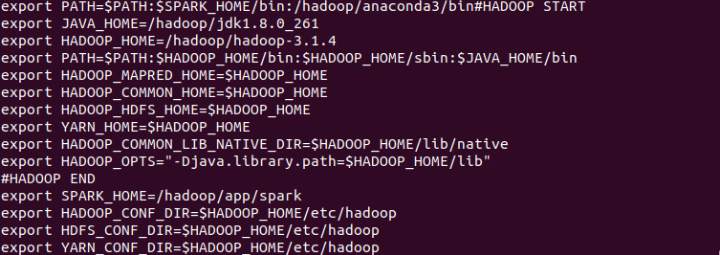


图2环境配置文件

在终端使用wget命令下载anaconda3安装包，输入以下命令开始下载：

**$ wget https://repo.anaconda.com/archive/ Anaconda3-2020.02-Linux-x86\_64.sh**

进入到安装包目录，执行命令安装anaconda3：

**$ cd /hadoop/anaconda3**

**$ bash Anaconda3-2020.02-Linux-x86\_64.sh**

安装结果如图3所示：

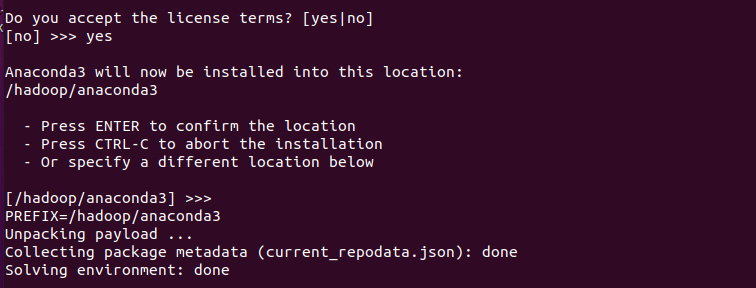


图3 anaconda3安装结果

安装好后设置环境变量（注意图2中在PATH中添加了anaconda3的bin目录）。

保存关闭后，执行以下命令使得环境变量生效：

**$ source /hadoop/.bash\_profile**

接下来配置PySpark driver，向/hadoop/.bash\_profile配置文件中添加以下语句：

**export PYSPARK\_DRIVER\_PYTHON=jupyter**

**export PYSPARK\_DRIVER\_PYTHON\_OPTS='notebook'**

重新在命令行输入pyspark，执行结果如图4所示：

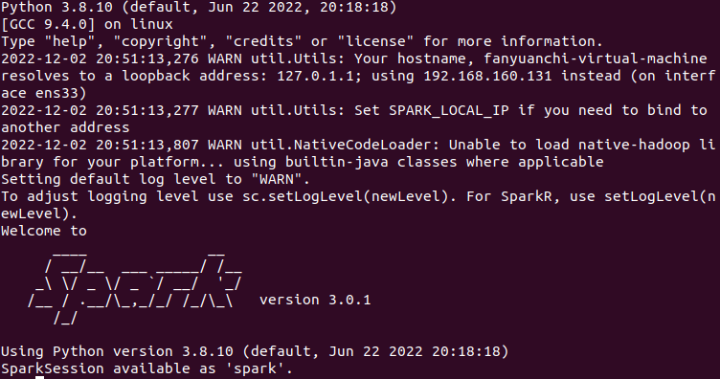


图4 pyspark执行结果

会自动启动jupyter，或者在浏览器输入提示url，开始创建python环境，就可以进行pyspark代码编写和运行了，创建结果如图5所示：

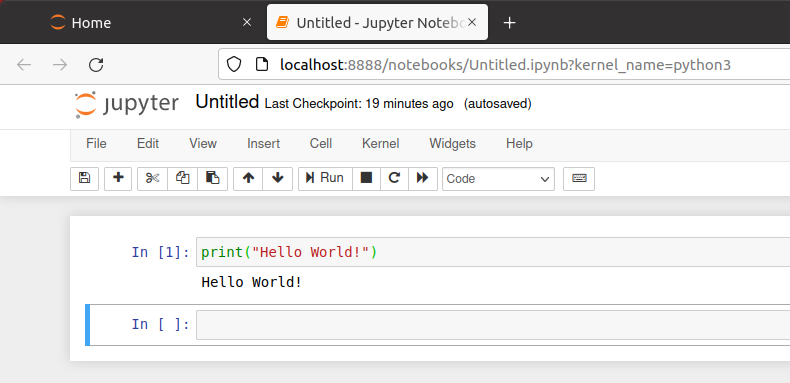


图5 pyspark代码编写和运行

1. **项目实践过程**
2. **查看飞机延误时间最长的前10名航班。**

实现代码如下：

代码1 查看飞机延误时间最长的前10名航班

import pandas as pd

df = pd.read\_csv(“/hadoop/data/DelayedFlights.csv”)

df = df.sort\_values(by=’DepDelay’,ascending=False).head(10)[ ‘DepDelay’]

print(df)

1. **计算延误的和没有延误的航空公司的比例。**

实现代码如下：

代码2 计算延误的和没有延误的航空公司的比例

import pandas as pd

df = pd.read\_csv(“/hadoop/data/DelayedFlights.csv”)

df\_undelay = df[df[‘ArrDelay’]<=0]

df\_delay = df[df[‘ArrDelay’]>0]

print(df\_undelay.count()/df\_delay.count())

1. **分析一天中延误最严重的飞行时间。**

实现代码如下（将数据上传至HDFS并使用scala实现）：

代码3 数据导入及格式处理

//调用sqlContext 提供的 read 接口，导入数据集

val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val flightData = sqlContext.read.format("com.databricks.spark.csv").option("header","true").load("/hangban/DelayedFlights.csv")

//将flightData 其注册为临时表

flightData.registerTempTable("flights")

代码4 分析一天中延误最严重的飞行时间

val queryFlightNumResult6 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 0 AND 600")

val queryFlightNumResult6 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 601 AND 1000")

val queryFlightNumResult6 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 1001 AND 1400")

val queryFlightNumResult6 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 1401 AND 1900")

val queryFlightNumResult6 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 1901 AND 2359")

1. **分析一周中延误最严重的飞行时间。**

实现代码如下（将数据上传至HDFS并使用scala实现）：

代码5 分析一周中延误最严重的飞行时间

val queryFlightNumResult13=sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayofWeek = 1")

val queryFlightNumResult13=sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayofWeek = 2")

val queryFlightNumResult13=sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayofWeek = 3")

val queryFlightNumResult13=sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayofWeek = 4")

val queryFlightNumResult13=sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayofWeek = 5")

val queryFlightNumResult13=sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayofWeek = 6")

val queryFlightNumResult13=sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayofWeek = 7")

1. **短途航班和长途航班，哪种航班取消更严重？**

实现代码如下：

代码6 分析短途航班和长途航班的取消情况

import pandas as pd

df = pd.read\_csv(“/hadoop/data/DelayedFlights.csv”)

df\_lc = df[(df[‘Distance’]>=1000)&(df[‘Cancelled’]==1)]

df\_lu = df[(df[‘Distance’]>=1000)&(df[‘Cancelled’]==0)]

df\_sc = df[(df[‘Distance’]<1000)&(df[‘Cancelled’]==1)]

df\_su = df[(df[‘Distance’]<1000)&(df[‘Cancelled’]==0)]

lr =df\_lc.count()/df\_lu.count()

sr =df\_sc.count()/df\_su.count()

print(“long distance: \n”,lr,”short distance:”,sr)

1. **建立机器学习算法模型，预测未来航班取消情况。**

实现代码如下：

代码7 数据导入及格式处理

import pandas as pd

def write\_dataset():

df = pd.read\_csv("D:/Pycharm/PythonProject/FlightDelay/DelayedFlights.csv")

df\_train = df[['Month', 'DayofMonth', 'CRSDepTime', 'FlightNum', 'CRSElapsedTime', 'Distance', 'Cancelled']]

df\_test = df\_train.sample(50000)

df\_train = df\_train.sort\_values(by=['Month', 'DayofMonth'], ascending=[True, True])

df\_test = df\_test.sort\_values(by=['Month', 'DayofMonth'], ascending=[True, True])

train\_res = df\_train['Cancelled']

train\_data = df\_train.drop(columns=['Cancelled'])

test\_res = df\_test['Cancelled']

test\_data = df\_test.drop(columns=['Cancelled'])

train\_res.to\_csv("D:/Pycharm/PythonProject/FlightDelay/trainRes.csv")

train\_data.to\_csv("D:/Pycharm/PythonProject/FlightDelay/trainData.csv")

test\_res.to\_csv("D:/Pycharm/PythonProject/FlightDelay/testRes.csv")

test\_data.to\_csv("D:/Pycharm/PythonProject/FlightDelay/testData.csv")

pass

代码8 数据结构规约定义及数据装载

import pandas as pd

import numpy as np

import keras

from keras.models import Sequential

from keras.layers import Dense, LSTM

from keras.optimizers import Adadelta

from matplotlib import pyplot as plt

batch\_size = 256

num\_classes = 2

epochs = 10

def load\_data():

trd = pd.read\_csv("D:/Pycharm/PythonProject/FlightDelay/trainData.csv")

trs = pd.read\_csv("D:/Pycharm/PythonProject/FlightDelay/trainRes.csv")

ted = pd.read\_csv("D:/Pycharm/PythonProject/FlightDelay/testData.csv")

tes = pd.read\_csv("D:/Pycharm/PythonProject/FlightDelay/testRes.csv")

trd = np.array(trd)

trs = np.array(trs)

ted = np.array(ted)

tes = np.array(tes)

trd = np.delete(trd, 0, axis=1)

trs = np.delete(trs, 0, axis=1)

ted = np.delete(ted, 0, axis=1)

tes = np.delete(tes, 0, axis=1)

trd = trd[:, :, np.newaxis]

ted = ted[:, :, np.newaxis]

trs = keras.utils.to\_categorical(trs, num\_classes)

tes = keras.utils.to\_categorical(tes, num\_classes)

print(trd.shape)

return (trd, trs), (ted, tes)

pass

代码9 训练及验证模型

def neural\_train(trd, trs, ted, tes):

model = Sequential()

# model.add(Dense(17, activation='relu', input\_shape=(23,)))

model.add(LSTM(29, return\_sequences=True))

# model.add(Dropout(0.2))

model.add(LSTM(23))

model.add(Dense(num\_classes, activation='softmax'))

model.compile(loss='categorical\_crossentropy',

optimizer=Adadelta(), metrics=['accuracy'])

his = model.fit(trd, trs, batch\_size=batch\_size, epochs=epochs, verbose=1,

validation\_data=(ted, tes))

model.save("D:/Pycharm/PythonProject/FlightDelay/model.h5")

return his

pass

def print\_history(train\_history):

# 绘制训练 & 验证(回归)的准确率值

plt.plot(train\_history.history['accuracy'])

plt.plot(train\_history.history['val\_accuracy'])

# 绘制训练 & 验证(回归)的损失率值

plt.plot(train\_history.history['loss'])

plt.plot(train\_history.history['val\_loss'])

plt.title('Model accuracy&loss')

plt.xlabel('Epoch')

plt.legend(['Train\_acc', 'Val\_acc', 'Train\_loss', 'Val\_loss'])

plt.show()

pass

1. **项目结果与分析（含重要数据结果分析或核心代码流程分析）**
2. **查看飞机延误时间最长的前10名航班。**

代码运行结果如图6所示：

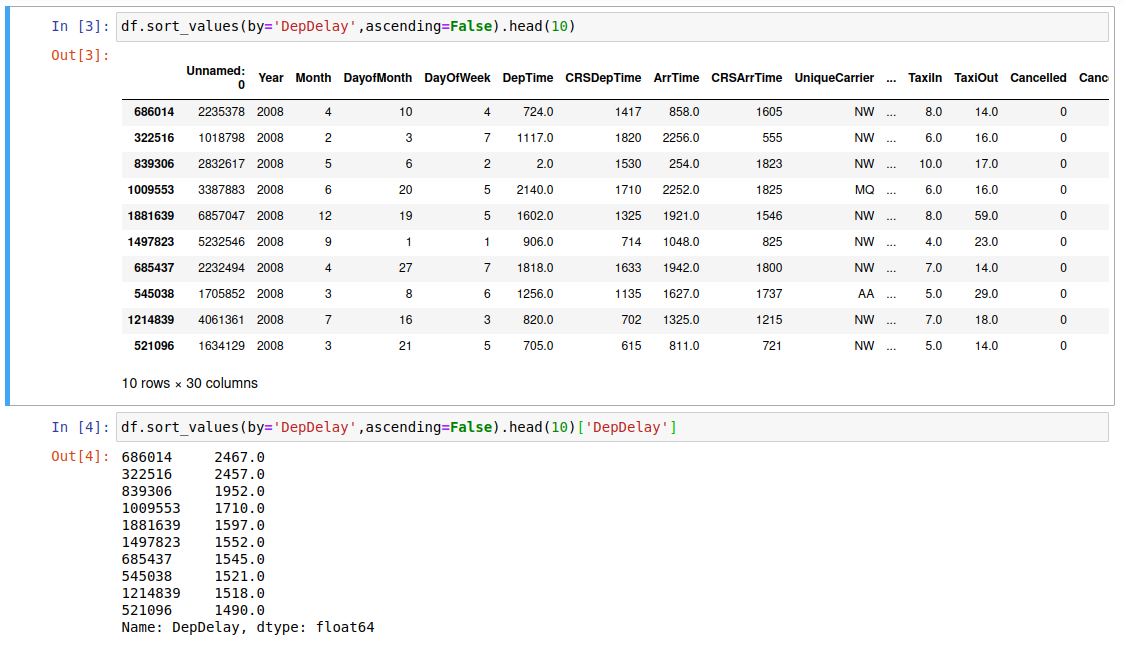


图6 查看飞机延误时间最长的前10名航班运行结果

如图6所示，延误时间有两种衡量标准——出发延误时间DepDelay以及抵达延误时间ArrDelay。这里以“出发延误时间”DepDelay作为衡量标准进行数据分析，由于数据项多长，针对具体延误时间另作显示，在具体航班下显示。

此外，还有以“抵达延误时间”ArrDelay作为衡量标准进行数据分析，同样因为数据项多长，针对具体延误时间另作显示，在具体航班下显示。执行结果如图7所示：



图7 查看飞机延误时间最长的前10名航班运行结果

1. **计算延误的和没有延误的航空公司的比例。**

代码运行结果如图8所示：

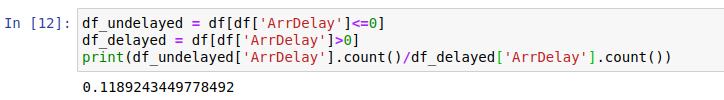


图8 延误的和没有延误的航空公司的比例运行结果

如图8所示，这里以“抵达延误时间”ArrDelay作为衡量标准进行数据分析。根据相关数据结构规约，ArrDelay数据项小于等于0视为没有延误；反之，ArrDelay数据项大于0视为有延误。因此对两种情况分别统计并计算二者比例。得到延误的和没有延误的航空公司的比例，约为11.89%。

1. **分析一天中延误最严重的飞行时间。**

代码运行结果如图9所示：

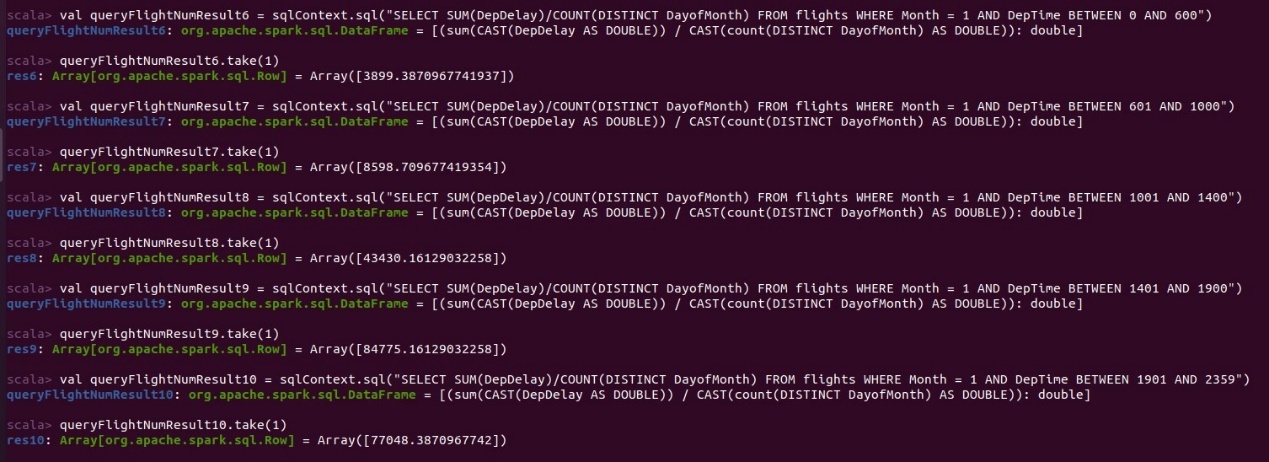


图9 一天中延误最严重的飞行时间分析结果

如图9所示，为了分时间段统计延误时长，我们将一天的时间分为五段：

凌晨（00:00 - 06:00）：大部分人在休息，我们合理假设该时间段内延误时长短；早上（06:01 - 10:00）：一些早班机会选择在此时间出发，延误时长变长；中午（10:01 - 14:00）：人们通常在这个时候方便抵达机场，在该时间段的延误时长可能增加；下午（14:01 - 19:00）：在下午出发更为方便，在该时间段的延误时长可能继续增加；晚上（19:01 - 23:59）：接近凌晨的航班数量可能会更少，但由于能见度原因，延误时长可能仍然较高。

有了上述准备，我们用关键字 `BETWEEN x AND y` 来设置数据的起止范围，用`SUM`函数对 DepDelay(出发延误时间) 进行统计，即 `SUM(DepDelay)`，使用`COUNT(DISTINCT DayofMonth)` 计算每个月的天数，最后计算出每天某时段的平均延误时长。这里选择的时间段为00:00 - 06:00，查询得到该时段的平均延误时长为 3899.38分钟。

以此类推，依次统计00:00 - 06:00；06:01 - 10:00；10:01 - 14:00；14:01 - 19:00；19:01 - 23:59时段的平均延误时长，得到结果。

最终统计的结果表明：2008年1月，每天延误最严重的时段为下午。该时段的平均延误时长为 84775.16分钟。

1. **分析一周中延误最严重的飞行时间。**

代码运行结果如图10所示：

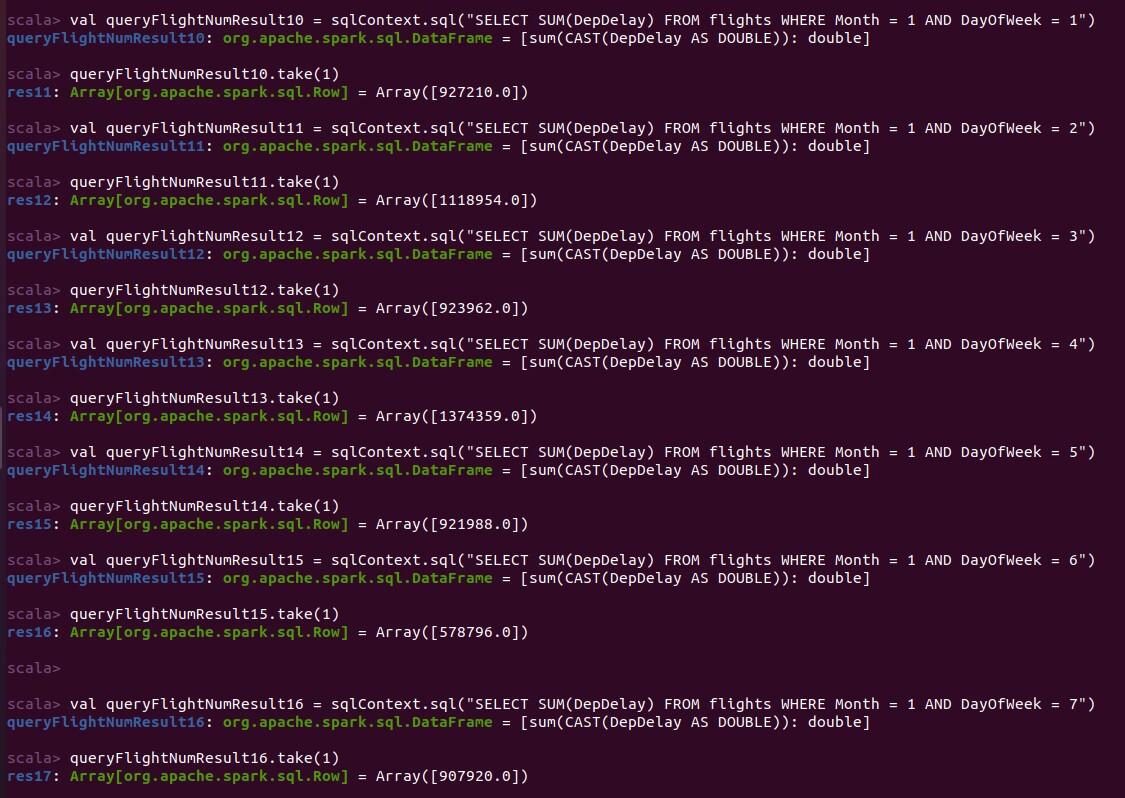


图10 一周中延误最严重的飞行时间分析结果

如图10所示，分别统计2008年一月份星期一到星期日的延误时间总和。最后得到结果2008年一月份星期四延误最严重，共计延误时间1374359分钟。

**5. 短途航班和长途航班，哪种航班取消更严重？**

代码运行结果如图11所示：

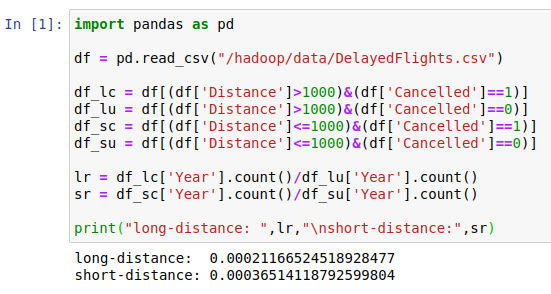


图11 短途航班和长途航班取消情况分析

如图11所示，这里以航行距离Distance为衡量标准进行数据分析。航程大于1000英里认为是长途航班，航程小于等于1000英里认为是短途航班。根据取消情况Cancelled（1代表取消，0代表正常）对长途航班取消、长途航班未取消、短途航班取消以及短途航班未取消四种情况分别统计并计算长途航班、短途航班的取消比例。

程序运行结果：长途航班取消比例约为0.02%，短途航班取消比例约为0.04%。综合结果来看短途航班比长途航班取消更严重。

1. **建立机器学习算法模型，预测未来航班取消情况。**

Long ShortTerm 网络——一般就叫做LSTM——是一种RNN特殊的类型，可以学习长期依赖信息。当然，LSTM和基线RNN并没有特别大的结构不同，但是它们用了不同的函数来计算隐状态。

LSTM的“记忆”叫做细胞/cells，可以直接把它们想做黑盒，这个黑盒的输入为前状态h\_{t-1}和当前输入x\_{t}。这些“细胞”会决定哪些之前的信息和状态需要保留/记住，而哪些要被抹去。实际的应用中发现，这种方式可以有效地保存很长时间之前的关联信息。

数据清晰规约，由于本模块用于预测航班取消情况，因此需要筛选航班起飞前可以确定的数据，这里选择月份Month，日期DayofMonth，计划起飞时间CRSDepTime，航班号FlightNum，预计到达时间CRSElapsedTime以及航程举例Distance。并将航班取消情况Cancelled作为数据集标签。

接下来搭建神经网络，神经网络模型结构如图12所示：

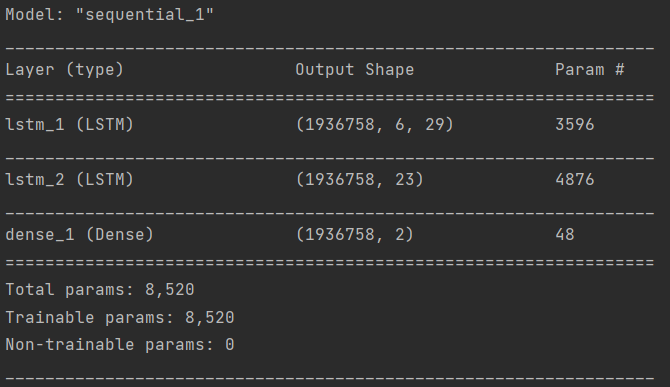


图12 神经网络模型结构

如图13所示，本模型为分类模型，由3层神经网络层组成。第一层为LSTM神经层，拥有29个LSTM细胞；第二层同样为LSTM神经层，拥有23个LSTM细胞；第三层为全连接层，用于输出分类结果，激活函数为“softmax”。

深度学习神经网络训练结果如图13所示：

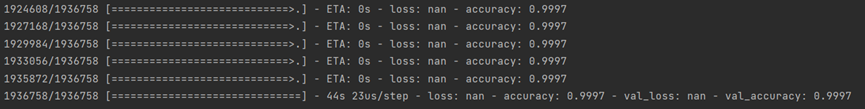


图13深度学习神经网络训练结果

如图13所示，神经网络训练准确率99.97%，分析原因在于取消航班占所有航班比例过小，导致即使训练世代仅为1轮，神经网络也会达到“过拟合”状态。

神经网络训练过程记录可视化表示如图14所示：

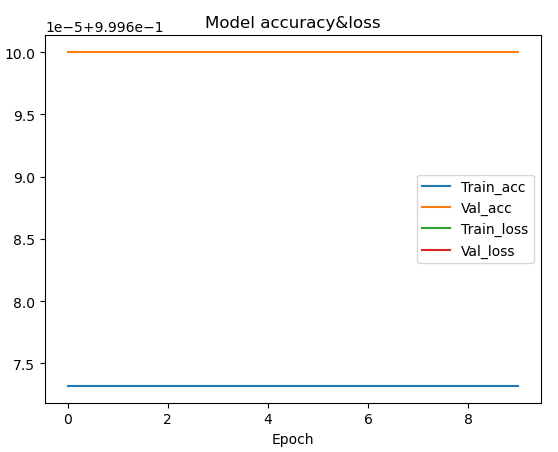


图14 神经网络训练过程记录可视化

**十一、总结及心得体会**

通过本次实验，我掌握了使用PySpark来分析数据的能力。并进一步掌握基于Python语言进行Spark Application编程，完成数据获取、处理、数据分析及可视化方面常用的数据分析方法与技巧。

同时，我也学习了很多新知识，如长短期记忆神经网络LSTM的原理与结构，深度学习神经网络的搭建方法与技巧。

最后，通过实践，我复习了课堂上所学的有关Spark内存计算、流计算、图计算与部分机器学习模型算法的有关知识，这对于我日后的深入研究有较大的帮助。

**十二、对本项目过程及方法、手段的改进建议**

如果在老师又是时间的情况下可以让实验指导手册提供一些常见报错处理方法或相关提问论坛。自行查询信息熵过大有可能造成操作系统损坏。

其他方面无重大缺陷，操作指导与实验介绍详细，操作流程可视化使实验过程可以有参考地进行，这一点值得提倡。