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 罗瑜

实验教师 罗瑜

学生信息：

序号	学号	姓名
1	2020***	***
2	***	***
3	***	***
4	***	***
5	***	***

电子科技大学教务处制表

电 子 科 技 大 学

项 目 报 告

指导教师： 罗瑜　　**地点：**第二教学楼 208

一、 项目名称：航空公司延误和取消分析项目

二、 项目时间：2023.12.20—2023.12.30

三、 项目原理

Spark 是专为大规模数据处理而设计的快速通用的计算引擎。Spark，拥有 Hadoop MapReduce 所具有的优点;但不同于 MapReduce 的是--Job 中间输出结果可以保存在内存中，从而不再需要读写 HDFS，因此 Spark 能更好地适用于数据挖掘与机器学习等需要迭代的 MapReduce 的算法；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其中一个例子就是 spark 可以运行在 hadoop 集群上。

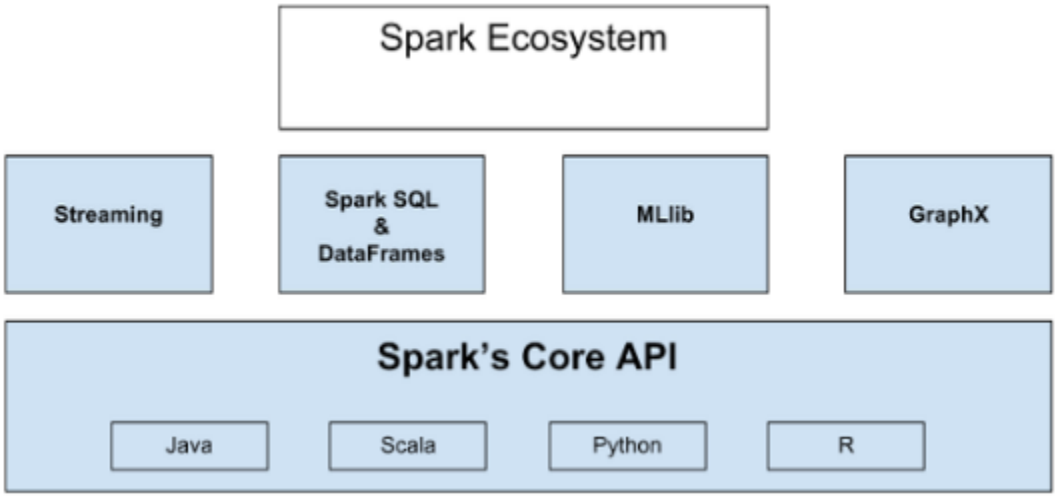


图 1 Apache Spark 应用架构图

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。

四、 项目内容

1. 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）；
2. 本机安装并测试 Jupyter Notebook；
3. 运行启动 PySpark 对航班延误数据集进行分析计算，并对新的输入做出预测。

五、 需求分析与设计

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。读入航班数据并处理（可以存入 HDFS、HBase 等），制作图形和表格来进行数据分析，建立预测模型，预测航班取消情况，分析至少包括以下内容：

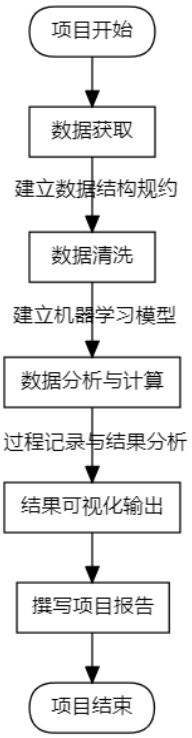
1. 查看飞机延误时间最长的前 10 名航班；
2. 计算延误的和没有延误的航空公司的比例；
3. 分析一天中、一周中延误最严重的飞行时间；
4. 分析短途航班和长途航班，哪种航班取消更严重？；
5. 建立机器学习算法模型，预测未来航班取消情况。

此外项目的非功能性需求包括以下内容：

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

六、 项目计划

项目开始后，首先获得航班延误数据集；然后对数据进行清洗得到有序的、干净的、符合接口所需结构的数据集；然后经过本小组讨论觉定选择 LSTM 长短期记忆神经网络模型进行分析训练；之后对得到的结果进行可视化展示；最后记录实验过程，完成实验结果分析报告。

七、 项目环境配置管理

7.1 操作系统： ubuntu-20.0.4、windows10、linux

7.2 开发工具：Pyspark，Jupyter Notebook，Pycharm，vmware

7.3 配置过程

确认 Spark 是否完成伪分布式配置（实验一要求的 spark 环境），使用 jps 命令查看 master 和 work 进程是否工作

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

然后输入 hadoop 用户的密码

$ 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：

```
export PATH=$PATH:$SPARK_HOME/bin:/hadoop/anaconda3/bin#HADOOP START
export JAVA_HOME=/hadoop/jdk1.8.0_261
export HADOOP_HOME=/hadoop/hadoop-3.1.4
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$JAVA_HOME/bin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
#HADOOP END
export SPARK_HOME=/hadoop/app/spark
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HDFS_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

图 2 环境配置文件

将 U 盘里拷贝的 anaconda 的 sh 脚本上传到 hadoop 文件夹下的 anaconda3 文件夹。然后进入到安装包目录，执行命令安装 anaconda3：

$ cd /hadoop/anaconda3

$ bash Anaconda3-2020.10-Linux-x86_64.sh

安装结果如图 3 所示：

```
Do you accept the license terms? [yes|no]
[no] >>> yes

Anaconda3 will now be installed into this location:
/hadoop/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/hadoop/anaconda3] >>>
PREFIX=/hadoop/anaconda3
Unpacking payload ...
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

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：

$ source /hadoop/.bash_profile

使更新系统的环境变量

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'
```

执行完成后再次使用 source 指令

八、 项目实践过程

1. **查看飞机延误时间最长的前 10 名航班。**

实现代码如下：

```
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)
```

图 1 查看飞机延误时间最长的前 10 名航班的代码

2. **计算延误的和没有延误的航空公司的比例。**

实现代码如下：

```
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())
```

图 2 计算延误的和没有延误的航空公司的比例代码

3. **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）：

```scala
//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")
```

图 3 数据导入及格式处理代码

```scala
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")
```

图 4 分析一天中延误最严重的飞行时间代码

4. 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）：

```scala
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")
```

图 5 分析一周中延误最严重的飞行时间代码

5. 短途航班和长途航班，哪种航班取消更严重？

实现代码如下：

```python
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)
```

图 6 分析短途航班和长途航班的取消情况代码

6. 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：

```
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")
```

图 7 数据导入及格式处理代码

```python
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)
```

图 8 数据结构规约定义及数据装载代码

```python
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


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()

load_data()
```

图 9 训练及验证模型代码

九、 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）

1. 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：

```
In [3]: df.sort_values(by='DepDelay',ascending=False).head(10)
```

Out[3]:

	Unnamed: 0	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	...	TaxiIn	TaxiOut	Cancelled	Canc
686014	2235378	2008	4	10	4	724.0	1417	858.0	1605	NW	...	8.0	14.0	0	
322516	1018798	2008	2	3	7	1117.0	1820	2256.0	555	NW	...	6.0	16.0	0	
839306	2832617	2008	5	6	2	2.0	1530	254.0	1823	NW	...	10.0	17.0	0	
1009553	3387883	2008	6	20	5	2140.0	1710	2252.0	1825	MQ	...	6.0	16.0	0	
1881639	6857047	2008	12	19	5	1602.0	1325	1921.0	1546	NW	...	8.0	59.0	0	
1497823	5232546	2008	9	1	1	906.0	714	1048.0	825	NW	...	4.0	23.0	0	
685437	2232494	2008	4	27	7	1818.0	1633	1942.0	1800	NW	...	7.0	14.0	0	
545038	1705852	2008	3	8	6	1256.0	1135	1627.0	1737	AA	...	5.0	29.0	0	
1214839	4061361	2008	7	16	3	820.0	702	1325.0	1215	NW	...	7.0	18.0	0	
521096	1634129	2008	3	21	5	705.0	615	811.0	721	NW	...	5.0	14.0	0	

10 rows × 30 columns

```
In [4]: df.sort_values(by='DepDelay',ascending=False).head(10)['DepDelay']
```

```
Out[4]: 686014     2467.0
322516     2457.0
839306     1952.0
1009553    1710.0
1881639    1597.0
1497823    1552.0
685437     1545.0
545038     1521.0
1214839    1518.0
521096     1490.0
Name: DepDelay, dtype: float64
```

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“DepDelay”和”ArrDelay”两种延误时间，前者叫做出发延误时间、后者叫做到达延误时间。这里以“出发延误时间”DepDelay 作为衡量标准进行数据分析。此外，还有以“抵达延误时间”ArrDelay 作为衡量标准进行数据分析。执行结果如图 7 所示：

```
In [6]: df.sort_values(by='ArrDelay',ascending=False).head(10)
```

Out[6]:

	Unnamed: 0	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	...	TaxiIn	TaxiOut	Cancelled	Canc
322516	1018798	2008	2	3	7	1117.0	1820	2256.0	555	NW	...	6.0	16.0	0	
686014	2235378	2008	4	10	4	724.0	1417	858.0	1605	NW	...	8.0	14.0	0	
839306	2832617	2008	5	6	2	2.0	1530	254.0	1823	NW	...	10.0	17.0	0	
1009553	3387883	2008	6	20	5	2140.0	1710	2252.0	1825	MQ	...	6.0	16.0	0	
1881639	6857047	2008	12	19	5	1602.0	1325	1921.0	1546	NW	...	8.0	59.0	0	
1497823	5232546	2008	9	1	1	906.0	714	1048.0	825	NW	...	4.0	23.0	0	
685437	2232494	2008	4	27	7	1818.0	1633	1942.0	1800	NW	...	7.0	14.0	0	
163379	527950	2008	1	5	6	800.0	1045	1452.0	1327	AA	...	14.0	71.0	0	
1214839	4061361	2008	7	16	3	820.0	702	1325.0	1215	NW	...	7.0	18.0	0	
521096	1634129	2008	3	21	5	705.0	615	811.0	721	NW	...	5.0	14.0	0	

10 rows × 30 columns

```
In [7]: df.sort_values(by='ArrDelay',ascending=False).head(10)['ArrDelay']
```

```
Out[7]: 322516     2461.0
686014     2453.0
839306     1951.0
1009553    1707.0
1881639    1655.0
1497823    1583.0
685437     1542.0
163379     1525.0
1214839    1510.0
521096     1490.0
Name: ArrDelay, dtype: float64
```

图 7 查看飞机延误时间最长的前 10 名航班运行结果

2. 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：

```
In [12]: df_undelayed = df[df['ArrDelay']<=0]
         df_delayed = df[df['ArrDelay']>0]
         print(df_undelayed['ArrDelay'].count()/df_delayed['ArrDelay'].count())

         0.1189243449778492
```

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%。

3. 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：

```
scala> val queryFlightNumResult6 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 0 AND 600")
queryFlightNumResult6: org.apache.spark.sql.DataFrame = [(sum(CAST(DepDelay AS DOUBLE)) / CAST(count(DISTINCT DayofMonth) AS DOUBLE)): double]

scala> queryFlightNumResult6.take(1)
res6: Array[org.apache.spark.sql.Row] = Array([3899.3870967741937])

scala> val queryFlightNumResult7 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 601 AND 1000")
queryFlightNumResult7: org.apache.spark.sql.DataFrame = [(sum(CAST(DepDelay AS DOUBLE)) / CAST(count(DISTINCT DayofMonth) AS DOUBLE)): double]

scala> queryFlightNumResult7.take(1)
res7: Array[org.apache.spark.sql.Row] = Array([8598.709677419354])

scala> val queryFlightNumResult8 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 1001 AND 1400")
queryFlightNumResult8: org.apache.spark.sql.DataFrame = [(sum(CAST(DepDelay AS DOUBLE)) / CAST(count(DISTINCT DayofMonth) AS DOUBLE)): double]

scala> queryFlightNumResult8.take(1)
res8: Array[org.apache.spark.sql.Row] = Array([43430.16129032258])

scala> val queryFlightNumResult9 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 1401 AND 1900")
queryFlightNumResult9: org.apache.spark.sql.DataFrame = [(sum(CAST(DepDelay AS DOUBLE)) / CAST(count(DISTINCT DayofMonth) AS DOUBLE)): double]

scala> queryFlightNumResult9.take(1)
res9: Array[org.apache.spark.sql.Row] = Array([84775.16129032258])

scala> val queryFlightNumResult10 = sqlContext.sql("SELECT SUM(DepDelay)/COUNT(DISTINCT DayofMonth) FROM flights WHERE Month = 1 AND DepTime BETWEEN 1901 AND 2359")
queryFlightNumResult10: org.apache.spark.sql.DataFrame = [(sum(CAST(DepDelay AS DOUBLE)) / CAST(count(DISTINCT DayofMonth) AS DOUBLE)): double]

scala> queryFlightNumResult10.take(1)
res10: Array[org.apache.spark.sql.Row] = Array([77048.3870967742])
```

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。

4. 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：

```
scala> val queryFlightNumResult10 = sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayOfWeek = 1")
queryFlightNumResult10: org.apache.spark.sql.DataFrame = [sum(CAST(DepDelay AS DOUBLE)): double]

scala> queryFlightNumResult10.take(1)
res11: Array[org.apache.spark.sql.Row] = Array([927210.0])

scala> val queryFlightNumResult11 = sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayOfWeek = 2")
queryFlightNumResult11: org.apache.spark.sql.DataFrame = [sum(CAST(DepDelay AS DOUBLE)): double]

scala> queryFlightNumResult11.take(1)
res12: Array[org.apache.spark.sql.Row] = Array([1118954.0])

scala> val queryFlightNumResult12 = sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayOfWeek = 3")
queryFlightNumResult12: org.apache.spark.sql.DataFrame = [sum(CAST(DepDelay AS DOUBLE)): double]

scala> queryFlightNumResult12.take(1)
res13: Array[org.apache.spark.sql.Row] = Array([923962.0])

scala> val queryFlightNumResult13 = sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayOfWeek = 4")
queryFlightNumResult13: org.apache.spark.sql.DataFrame = [sum(CAST(DepDelay AS DOUBLE)): double]

scala> queryFlightNumResult13.take(1)
res14: Array[org.apache.spark.sql.Row] = Array([1374359.0])

scala> val queryFlightNumResult14 = sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayOfWeek = 5")
queryFlightNumResult14: org.apache.spark.sql.DataFrame = [sum(CAST(DepDelay AS DOUBLE)): double]

scala> queryFlightNumResult14.take(1)
res15: Array[org.apache.spark.sql.Row] = Array([921988.0])

scala> val queryFlightNumResult15 = sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayOfWeek = 6")
queryFlightNumResult15: org.apache.spark.sql.DataFrame = [sum(CAST(DepDelay AS DOUBLE)): double]

scala> queryFlightNumResult15.take(1)
res16: Array[org.apache.spark.sql.Row] = Array([578796.0])

scala>

scala> val queryFlightNumResult16 = sqlContext.sql("SELECT SUM(DepDelay) FROM flights WHERE Month = 1 AND DayOfWeek = 7")
queryFlightNumResult16: org.apache.spark.sql.DataFrame = [sum(CAST(DepDelay AS DOUBLE)): double]

scala> queryFlightNumResult16.take(1)
res17: Array[org.apache.spark.sql.Row] = Array([907920.0])
```

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：

```
In [1]: 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['Year'].count()/df_lu['Year'].count()
sr = df_sc['Year'].count()/df_su['Year'].count()

print("long-distance: ",lr,"\nshort-distance:",sr)
```

```
long-distance:  0.00021166524518928477
short-distance: 0.00036514118792599804
```

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。

6. 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：

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (1936758, 6, 29)          3596
_________________________________________________________________
lstm_2 (LSTM)                (1936758, 23)             4876
_________________________________________________________________
dense_1 (Dense)              (1936758, 2)              48
=================================================================
Total params: 8,520
Trainable params: 8,520
Non-trainable params: 0
_________________________________________________________________
```

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：

```
1924608/1936758 [============================>.] - ETA: 0s - loss: nan - accuracy: 0.9997
1927168/1936758 [============================>.] - ETA: 0s - loss: nan - accuracy: 0.9997
1929984/1936758 [============================>.] - ETA: 0s - loss: nan - accuracy: 0.9997
1933056/1936758 [============================>.] - ETA: 0s - loss: nan - accuracy: 0.9997
1935872/1936758 [============================>.] - ETA: 0s - loss: nan - accuracy: 0.9997
1936758/1936758 [==============================] - 44s 23us/step - loss: nan - accuracy: 0.9997 - val_loss: nan - val_accuracy: 0.9997
```

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：

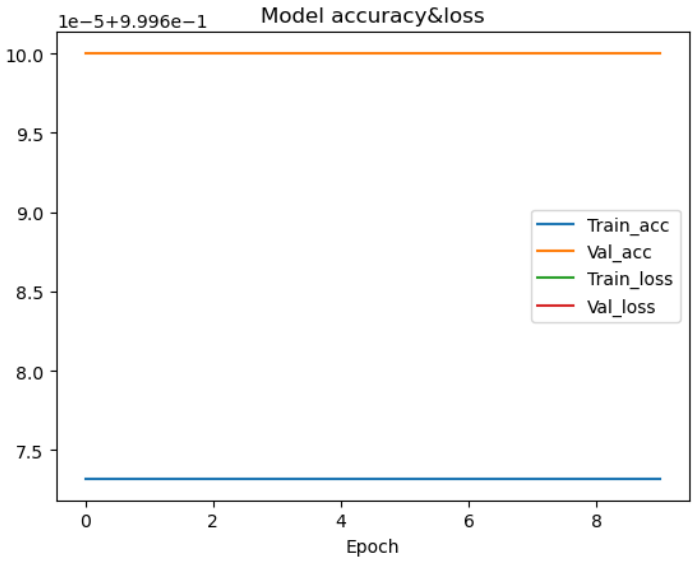


图 14 神经网络训练过程记录可视化

十一、总结及心得体会

通过本次实验，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，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，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

在共同完成一个项目时，应该充分了解和沟通，确保每一个人对项目的目标有一个清晰的认识，以便适当的改进方案，合理分配任务，确保大家的任务能够并发执行，提升团队合作效率。