Big_Data05

May 16, 2022

1 Name - Ambuj Mishra

2 Student ID - 202116003

This is a colab PDF containing both question codes and the description required in the assignment.

3 Spark Initialization

[ ]:
```
# innstall java
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# install spark (change the version number if needed)
!wget -q https://archive.apache.org/dist/spark/spark-3.0.0/spark-3.0.
 ↪0-bin-hadoop3.2.tgz

# unzip the spark file to the current folder
!tar xf spark-3.0.0-bin-hadoop3.2.tgz

# set your spark folder to your system path environment.
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"


# install findspark using pip
!pip install -q findspark
```

[ ]:
```
import findspark

findspark.init()
```

[ ]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder\
```

```
        .master("local")\
        .appName("Colab")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
```

[ ]:
```
type(spark)
```

[ ]:
```
pyspark.sql.session.SparkSession
```

[ ]:
```
sc=spark.sparkContext
```

4 Question-1

[ ]:
```
!head /content/2015-summary.csv
```

```
DEST_COUNTRY_NAME,ORIGIN_COUNTRY_NAME,count
United States,Romania,15
United States,Croatia,1
United States,Ireland,344
Egypt,United States,15
United States,India,62
United States,Singapore,1
United States,Grenada,62
Costa Rica,United States,588
Senegal,United States,40
```

[ ]:
```
flightData2015 = spark.read.option("inferSchema", "true").option("header",␣
↪"true").csv("/content/2015-summary.csv")
```

[ ]:
```
flightData2015.take(3)
```

[ ]:
```
[Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Romania',
count=15),
 Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Croatia', count=1),
 Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Ireland',
count=344)]
```

[ ]:
```
flightData2015.sort("count").explain()
```

```
== Physical Plan ==
*(1) Sort [count#18 ASC NULLS FIRST], true, 0
+- Exchange rangepartitioning(count#18 ASC NULLS FIRST, 200), true, [id=#32]
   +- FileScan csv [DEST_COUNTRY_NAME#16,ORIGIN_COUNTRY_NAME#17,count#18]
Batched: false, DataFilters: [], Format: CSV, Location:
InMemoryFileIndex[file:/content/2015-summary.csv], PartitionFilters: [],
```

```
PushedFilters: [], ReadSchema:
struct<DEST_COUNTRY_NAME:string,ORIGIN_COUNTRY_NAME:string,count:int>
```

[ ]:
```
spark.conf.set("spark.sql.shuffle.partitions", "5")
```

[ ]:
```
flightData2015.sort("count").take(2)
```

[ ]:
```
[Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Singapore',
count=1),
 Row(DEST_COUNTRY_NAME='Moldova', ORIGIN_COUNTRY_NAME='United States', count=1)]
```

4.1 DataFrames and SQL

[ ]:
```
flightData2015.createOrReplaceTempView("flight_data_2015")
```

[ ]:
```
sqlWay = spark.sql("""SELECT DEST_COUNTRY_NAME, count(1) FROM flight_data_2015␣
 ↪GROUP BY DEST_COUNTRY_NAME""")
dataFrameWay = flightData2015.groupBy("DEST_COUNTRY_NAME").count()

sqlWay.explain()
dataFrameWay.explain()
```

```
== Physical Plan ==
*(2) HashAggregate(keys=[DEST_COUNTRY_NAME#16], functions=[count(1)])
+- Exchange hashpartitioning(DEST_COUNTRY_NAME#16, 5), true, [id=#61]
   +- *(1) HashAggregate(keys=[DEST_COUNTRY_NAME#16],
functions=[partial_count(1)])
      +- FileScan csv [DEST_COUNTRY_NAME#16] Batched: false, DataFilters: [],
Format: CSV, Location: InMemoryFileIndex[file:/content/2015-summary.csv],
PartitionFilters: [], PushedFilters: [], ReadSchema:
struct<DEST_COUNTRY_NAME:string>


== Physical Plan ==
*(2) HashAggregate(keys=[DEST_COUNTRY_NAME#16], functions=[count(1)])
+- Exchange hashpartitioning(DEST_COUNTRY_NAME#16, 5), true, [id=#80]
   +- *(1) HashAggregate(keys=[DEST_COUNTRY_NAME#16],
functions=[partial_count(1)])
      +- FileScan csv [DEST_COUNTRY_NAME#16] Batched: false, DataFilters: [],
Format: CSV, Location: InMemoryFileIndex[file:/content/2015-summary.csv],
PartitionFilters: [], PushedFilters: [], ReadSchema:
struct<DEST_COUNTRY_NAME:string>
```

[ ]:
```
spark.sql("SELECT max(count) from flight_data_2015").take(1)
```

[ ]:
```
[Row(max(count)=370002)]
```

[ ]:
```
from pyspark.sql.functions import max

flightData2015.select(max("count")).take(1)
```

[ ]:
```
[Row(max(count)=370002)]
```

[ ]:
```
maxSql = spark.sql("""SELECT DEST_COUNTRY_NAME, sum(count) as destination_total␣
 ↪FROM flight_data_2015 GROUP BY DEST_COUNTRY_NAME ORDER BY sum(count) DESC␣
 ↪LIMIT 5""")
maxSql.show()
```

```
+-----------------+-----------------+
|DEST_COUNTRY_NAME|destination_total|
+-----------------+-----------------+
|    United States|           411352|
|           Canada|             8399|
|           Mexico|             7140|
|   United Kingdom|             2025|
|            Japan|             1548|
+-----------------+-----------------+
```

[ ]:
```
from pyspark.sql.functions import desc
flightData2015\
.groupBy("DEST_COUNTRY_NAME")\
.sum("count")\
.withColumnRenamed("sum(count)", "destination_total")\
.sort(desc("destination_total"))\
.limit(5)\
.show()
```

```
+-----------------+-----------------+
|DEST_COUNTRY_NAME|destination_total|
+-----------------+-----------------+
|    United States|           411352|
|           Canada|             8399|
|           Mexico|             7140|
|   United Kingdom|             2025|
|            Japan|             1548|
+-----------------+-----------------+
```

[ ]:
```
flightData2015\
.groupBy("DEST_COUNTRY_NAME")\
.sum("count")\
.withColumnRenamed("sum(count)", "destination_total")\
.sort(desc("destination_total"))\
.limit(5)\
.explain()
```

```
== Physical Plan ==
TakeOrderedAndProject(limit=5, orderBy=[destination_total#104L DESC NULLS LAST],
output=[DEST_COUNTRY_NAME#16,destination_total#104L])
+- *(2) HashAggregate(keys=[DEST_COUNTRY_NAME#16], functions=[sum(cast(count#18
as bigint))])
   +- Exchange hashpartitioning(DEST_COUNTRY_NAME#16, 5), true, [id=#227]
      +- *(1) HashAggregate(keys=[DEST_COUNTRY_NAME#16],
functions=[partial_sum(cast(count#18 as bigint))])
         +- FileScan csv [DEST_COUNTRY_NAME#16,count#18] Batched: false,
DataFilters: [], Format: CSV, Location:
InMemoryFileIndex[file:/content/2015-summary.csv], PartitionFilters: [],
PushedFilters: [], ReadSchema: struct<DEST_COUNTRY_NAME:string,count:int>
```

4.2 Physical Plan-

Physical Plan is an internal enhancement or optimization for Spark. It is generated after creation of the Optimized Logical Plan. Physical Plan is limited to Spark operation and for this, it will do an evaluation of multiple physical plans and finalize the suitable optimal physical plan. And ultimately, the finest Physical Plan runs. Once the finest Physical Plan is selected, executable code (DAG of RDDs) for the query is created which needs to be executed in a distributed manner on the cluster.

5 Question-2

5.1 Basic Query Example

[ ]:
```
# Import Spark SQL
from pyspark.sql import HiveContext, Row
# Or if you can't include the hive requirements
from pyspark.sql import SQLContext, Row
```

[ ]:
```
hiveCtx = HiveContext(sc)
```

[ ]:
```
input = hiveCtx.read.json("/content/testweet.json")
input.show()
```

```
# Register the input schema RDD
input.registerTempTable("tweets")
# Select tweets based on the retweetCount
topTweets = hiveCtx.sql("""SELECT text, retweetCount FROM tweets ORDER BY␣
 ↪retweetCount LIMIT 10""")
topTweets.collect()
```

```
+--------------+--------------------+--------------------+--------------+-----
------------+-----------------+--------------+-----------+--------------------+
-----------+-------------+------------+--------------------+--------------------
+-----------+--------------------+-------------------+
|contributorsIDs|           createdAt|currentUserRetweetId|hashtagEntities|
id|inReplyToStatusId|inReplyToUserId|isFavorited|isPossiblySensitive|isTruncated
|mediaEntities|retweetCount|              source|
text|urlEntities|                user|userMentionEntities|
+--------------+--------------------+--------------------+--------------+-----
------------+-----------------+--------------+-----------+--------------------+
-----------+-------------+------------+--------------------+--------------------
+-----------+--------------------+-------------------+
|             []|Nov 4, 2014 4:56:…|                  -1|
[]|529799371026485248|               -1|             -1|      false|
false|      false|           []|           0|<a href="http://t…|Adventures
With C…|         []|[Aug 5, 2011 9:42…|                  []|
+--------------+--------------------+--------------------+--------------+-----
------------+-----------------+--------------+-----------+--------------------+
-----------+-------------+------------+--------------------+--------------------
+-----------+--------------------+-------------------+
```

```
[ ]: [Row(text='Adventures With Coffee, Code, and Writing.', retweetCount=0)]
```

5.2 Spark SQL UDF

```
[ ]: from pyspark.sql.types import IntegerType

# Make a UDF to tell us how long some text is
hiveCtx.registerFunction("strLenPython", lambda x: len(x), IntegerType())
lengthSchemaRDD = hiveCtx.sql("SELECT strLenPython('text') FROM tweets LIMIT␣
 ↪10")
lengthSchemaRDD.collect()
```

```
[ ]: [Row(strLenPython(text)=4)]
```

5.3 Example 9-40

```
[ ]: multipleSumRDD = hiveCtx.sql("SELECT SUM(user.favouritesCount),␣
      ↪SUM(retweetCount), user.id FROM tweets GROUP BY user.id")
     multipleSumRDD.collect()
```

```
[ ]: [Row(sum(user.favouritesCount AS `favouritesCount`)=1095, sum(retweetCount)=0,
     id=15594928)]
```

5.4 Ideas about Sentiment Analysis in Spark

Sentiment Analysis in Spark is doable and feasible because of Spark's feature of live data handling and even if we want to perform offline stored tweet's sentiment analysis, still Spark provides proper platform for that.

5.5 Steps for Sentiment Analysis

Following steps are required if we want to perform sentiment analysis in Spark-

1. Data Preprocessing-

We need to clean our tweets and remove unnecessary information. For that we'll first tokenize our dataset and create tokens out of it. On the basis of that, we'll remove stopwords and not required tags and symbols.

2. Stemming or Lemmatization-

We will convert our tokens into their stem form or lemma form to equate the same stem/lemma words.

3. Vectorize dataset-

On the basis on the filtered data, we'll generate the vectors for every tweet.

4. Sentiment Classification-

On the basis of our requirement, we'll generate classes of dataset and based on the vectorized data, we'll train the model using some classifcation algorithm. This model will further help in generating sentiment results.

6 Question-3

6.1 Example : Spam Classification

```
[ ]: from pyspark.mllib.regression import LabeledPoint
     from pyspark.mllib.feature import HashingTF
     from pyspark.mllib.classification import LogisticRegressionWithSGD
```

```
spam = sc.textFile("spam.txt")
normal = sc.textFile("normal.txt")

# Create a HashingTF instance to map email text to vectors of 150 features.
tf = HashingTF(numFeatures = 150)

# Each email is split into words, and each word is mapped to one feature.
spamFeatures = spam.map(lambda email: tf.transform(email.split(" ")))
normalFeatures = normal.map(lambda email: tf.transform(email.split(" ")))

# Create LabeledPoint datasets for positive (spam) and negative (normal)␣
 ↪examples.
positiveExamples = spamFeatures.map(lambda features: LabeledPoint(1, features))
negativeExamples = normalFeatures.map(lambda features: LabeledPoint(0,␣
 ↪features))
trainingData = positiveExamples.union(negativeExamples)
trainingData.cache() # Cache since Logistic Regression is an iterative␣
 ↪algorithm.

# Run Logistic Regression using the SGD algorithm.
model = LogisticRegressionWithSGD.train(trainingData)

# Test on a positive example (spam) and a negative one (normal). We first apply
# the same HashingTF feature transformation to get vectors, then apply the␣
 ↪model.
posTest = tf.transform("O M G GET cheap stuff by sending money to ...".split("␣
 ↪"))
negTest = tf.transform("Hi Dad, I started studying Spark the other ...".split("␣
 ↪"))

print("Prediction for positive test example: {}".format(model.predict(posTest)))
print("Prediction for negative test example: {}".format(model.predict(negTest)))
```

```
Prediction for positive test example: 1
Prediction for negative test example: 0
```

6.2 Hashing TF

Using hashing TF, we generate hash code of any object which is irreversible in nature. Since, hashing is not reversible, you cannot restore original input from a hash vector. On the other hand, count vector with model (index) can be used to restore unordered input. As a consequence models created using hashed input can be much harder to interpret and monitor.

6.3 Logistic Regression

Logistic Regression in Spark can be implemented using *org.apache.spark.ml.classification.LogisticRegression* module.

In case of binary classification using logisticRegression, we assign first class as '0' and other one as '1'. We further try to calculate the probility of belongingness in class 1(y==1) given a sample 'x'.

P(y==1/x) = 1/ (1+e^-x)

Based on the probability, we generate the final discriminant function.

G(x) = G1(x) - G0(x)

Now, we can say if it is greater than 0, then the given sample will belong to class 1 otherwise in class 0.

In case of multiclass classification, we use one vs rest or pairwise approach to discriminate.

7 THANK YOU!

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('Big_Data05.ipynb')
```

```
--2022-05-16 23:45:54--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.110.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: 'colab_pdf.py'

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2022-05-16 23:45:54 (38.0 MB/s) - 'colab_pdf.py' saved [1864/1864]


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

```
Extracting templates from packages: 100%
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/Big_Data05.ipynb to pdf
[NbConvertApp] Writing 53644 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 73034 bytes to /content/drive/My Drive/Big_Data05.pdf
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

[2]:
```
'File ready to be Downloaded and Saved to Drive'
```

[ ]:
```

```