MD FAKRUL ISLAM(613839) BIG DATA TECHNOLOGY

https://github.com/aifakrul/BigDataTechnology-CSE522

Twitter Spark Streaming – 1st Exercise https://github.com/aifakrul/BigDataTechnology-CSE522/tree/main/SparkStreaming

I will show how to build a simple application that reads online streams from Twitter using Python, then processes the tweets using Apache Spark Streaming to identify hashtags.

Building the Twitter HTTP Client and Spark Streaming to process

```
def get_tweets():
        url = 'https://stream.twitter.com/1.1/statuses/filter.json'
        query_data = [('language', 'en'), ('locations', '-130,-20,100,50'),('track','#')]
        query_url = url + '?' + '&'.join([str(t[0]) + '=' + str(t[1]) for t in query_data])
        response = requests.get(query_url, auth=my_auth, stream=True)
        print(query_url, response)
        return response
```

```
def send_tweets_to_spark(http_resp, tcp_connection):
        for line in http_resp.iter_lines():
        try:
                full_tweet = json.loads(line)
                tweet_text = full_tweet['text']
                print("Tweet Text: " + tweet_text)
                print ("------------------------------------------")
                tcp_connection.send(tweet_text + '\n')
        except:
                e = sys.exc_info()[0]
                print("Error: %s" % e)
```

SPARK PROCESSING

```
ssc.start()
# wait for the streaming to finish
ssc.awaitTermination()

def aggregate_tags_count(new_values, total_sum):
        return sum(new_values) + (total_sum or 0)


def get_sql_context_instance(spark_context):
        if ('sqlContextSingletonInstance' not in globals()):
        globals()['sqlContextSingletonInstance'] = SQLContext(spark_context)
        return globals()['sqlContextSingletonInstance']
def process_rdd(time, rdd):
        print("----------- %s -----------" % str(time))
        try:
        # Get spark sql singleton context from the current context
        sql_context = get_sql_context_instance(rdd.context)
        # convert the RDD to Row RDD
        row_rdd = rdd.map(lambda w: Row(hashtag=w[0], hashtag_count=w[1]))
        # create a DF from the Row RDD
        hashtags_df = sql_context.createDataFrame(row_rdd)
        # Register the dataframe as table
        hashtags_df.registerTempTable("hashtags")
        # get the top 10 hashtags from the table using SQL and print them
        hashtag_counts_df = sql_context.sql("select hashtag, hashtag_count from hashtags order by hashtag_count desc limit 10")
        hashtag_counts_df.show()
        # call this method to prepare top 10 hashtags DF and send them
        send_df_to_dashboard(hashtag_counts_df)
        except:
        e = sys.exc_info()[0]
        print("Error: %s" % e)
```

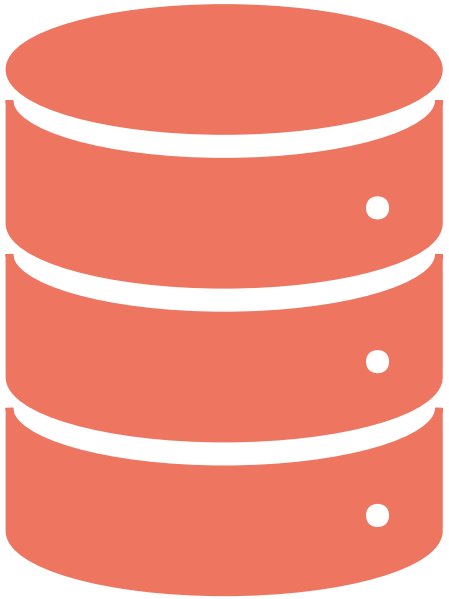
Twitter KAFKA SPARK HIVE Streaming – Second Exercise

https://github.com/aifakrul/BigDataTechnology-CSE522/tree/main/KafkaSparkHiveIntegration

In this example, I will do the below things.

- create a stream of tweets that will be sent to a Kafka queue
- pull the tweets from the Kafka cluster
- calculate the character count and word count for each tweet
- save this data to a Hive table

1.VM setup in my azure account
2.Install Kafka
3.Install Hadoop
4.Install Hive
5.Install Spark



KAFKA BROKER - PRODUCER

```
consumer_key = conf.consumer_key
consumer_secret_key = conf.consumer_secret_key

access_token = conf.access_token
access_token_secret = conf.access_token_secret

auth = tweepy.OAuthHandler(consumer_key, consumer_secret_key)
auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth)

streamer = Streamer()
stream = tweepy.Stream(auth=api.auth, listener=streamer)

try:
    producer = KafkaProducer(bootstrap_servers=KAFKA_BROKER)
except Exception as e:
    print(f'Error Connecting to Kafka --> {e}')
    sys.exit(1)
```

Fake Tweet Stream To KAFKA

```python
from kafka import KafkaProducer
from random import randint
from time import sleep
import sys

BROKER = 'localhost:9092'
TOPIC = 'tweets'

WORD_FILE = '/usr/share/dict/words'
WORDS = open(WORD_FILE).read().splitlines()

try:
    p = KafkaProducer(bootstrap_servers=BROKER)
except Exception as e:
    print(f"ERROR --> {e}")
    sys.exit(1)

while True:
    message = ''
    for _ in range(randint(2, 7)):
        message += WORDS[randint(0, len(WORDS)-1)] + ' '
    print(f">>> '{message}'")
    p.send(TOPIC, bytes(message, encoding="utf8"))
    sleep(randint(1,4))
```

SPARK PROCESSING – HIVE STORING

```
def handle_rdd(rdd):
    if not rdd.isEmpty():
        global ss
        df = ss.createDataFrame(rdd, schema=['text', 'words', 'length'])
        df.show()
        df.write.saveAsTable(name='default.tweets', format='hive', mode='append')

sc = SparkContext(appName="Something")
ssc = StreamingContext(sc, 5)

ss = SparkSession.builder \
        .appName("Something") \
        .config("spark.sql.warehouse.dir", "/user/hive/warehouse") \
        .config("hive.metastore.uris", "thrift://localhost:9083") \
        .enableHiveSupport() \
        .getOrCreate()

ss.sparkContext.setLogLevel('WARN')

ks = KafkaUtils.createDirectStream(ssc, ['tweets'], {'metadata.broker.list': 'localhost:9092'})

lines = ks.map(lambda x: x[1])

transform = lines.map(lambda tweet: (tweet, int(len(tweet.split())), int(len(tweet))))

transform.foreachRDD(handle_rdd)
```

SparkML using pyspark for Regression – Third Exercise

https://github.com/aifakrul/BigDataTechnology-CSE522/tree/main/ResearchProject

The goal of this exercise is predicting the housing prices by the given features. Let's predict the prices of the Boston Housing dataset by considering MEDV as the output variable and all the other variables as input. The whole exercise is done in Google Collab

INSTALL ALL LIBRARIES IN COLAB

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist/spark/spark-2.4.1/spark-2.4.1-bin-hadoop2.7.tgz
!tar xf spark-2.4.1-bin-hadoop2.7.tgz
!pip install -q findspark
```

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.3.2-bin-hadoop2.7"
```

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
```

LOAD BOSTON HOUSING DATASET

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression

dataset = spark.read.csv('BostonHousing.csv',inferSchema=True, header =True)
```

```
dataset.printSchema()
```

```
root
 |-- crim: double (nullable = true)
 |-- zn: double (nullable = true)
 |-- indus: double (nullable = true)
 |-- chas: integer (nullable = true)
 |-- nox: double (nullable = true)
 |-- rm: double (nullable = true)
 |-- age: double (nullable = true)
 |-- dis: double (nullable = true)
 |-- rad: integer (nullable = true)
 |-- tax: integer (nullable = true)
 |-- ptratio: double (nullable = true)
 |-- b: double (nullable = true)
```

DRIVE REGRESSION FOR PREDICTION

```
#Split training and testing data
train_data,test_data = finalized_data.randomSplit([0.8,0.2])


regressor = LinearRegression(featuresCol = 'Attributes', labelCol = 'medv')

#Learn to fit the model from training set
regressor = regressor.fit(train_data)

#To predict the prices on testing set
pred = regressor.evaluate(test_data)

#Predict the model
pred.predictions.show()
```

```
+--------------------+----+------------------+
|          Attributes|medv|        prediction|
+--------------------+----+------------------+
|[0.01301,35.0,1.5...|32.7| 30.07670363535312|
|[0.01538,90.0,3.7...|44.0| 37.75244575519337|
|[0.01778,95.0,1.4...|32.9|30.596108327253294|
|[0.0187,85.0,4.15...|23.1|25.717620889129734|
|[0.01965,80.0,1.7...|20.1|19.992379582220035|
|[0.02729,0.0,7.07...|34.7|30.425294527192754|
|[0.03113,0.0,4.39...|17.5|16.330496893793097|
|[0.03237,0.0,2.18...|33.4|28.578543755284294|
|[0.03306,0.0,5.19...|20.6| 22.16010760013387|
|[0.03359,75.0,2.9...|34.9| 34.42265990782376|
|[0.03537,34.0,6.0...|22.0|28.784081950984906|
|[0.03584,80.0,3.3...|23.5| 30.77179427151925|
|[0.03738,0.0,5.19...|20.7| 21.65956978285279|
|[0.04297,52.5,5.3...|24.8|26.706348196385573|
|[0.0456,0.0,13.89...|23.3|26.369847201011538|
|[0.04684,0.0,3.41...|22.6|26.949731074397704|
|[0.04981,21.0,5.6...|23.4| 23.90871028835852|
|[0.05372,0.0,13.9...|27.1|27.156639422924407|
|[0.05425,0.0,4.05...|24.6| 29.54769429196901|
|[0.06466,70.0,2.2...|22.5|29.459287514682245|
+--------------------+----+------------------+
```

```
#coefficient of the regression model
coeff = regressor.coefficients

#X and Y intercept
intr = regressor.intercept

print ("The coefficient of the model is : %a" %coeff)
print ("The Intercept of the model is : %f" %intr)
```

```
The coefficient of the model is : DenseVector([-0.1239, 0.056, 0.0205, 2.7283, -16.8634, 3.218, 0.0163, -1.4331, 0.3657, -0.0134, -0.9328, 0.009
6229])
The Intercept of the model is : 39.049826
```

Basic Statistical Analysis

I am done with the basic linear regression operation, i can go a bit further and analyze our model statistically by importing RegressionEvaluator module from Pyspark.

```
from pyspark.ml.evaluation import RegressionEvaluator
eval = RegressionEvaluator(labelCol="medv", predictionCol="prediction", metricName="rmse")

# Root Mean Square Error
rmse = eval.evaluate(pred.predictions)
print("RMSE: %.3f" % rmse)

# Mean Square Error
mse = eval.evaluate(pred.predictions, {eval.metricName: "mse"})
print("MSE: %.3f" % mse)

# Mean Absolute Error
mae = eval.evaluate(pred.predictions, {eval.metricName: "mae"})
print("MAE: %.3f" % mae)

# r2 - coefficient of determination
r2 = eval.evaluate(pred.predictions, {eval.metricName: "r2"})
print("r2: %.3f" %r2)
```

```
RMSE: 4.703
MSE: 22.118
```

THANK YOU