**Practical Exam (13/11/2021)**

**Practical 2**

**Aim:**

To develop a Map Reduce application and implement a program that analyzes electrical consumption data.

**Input:**

**sample.txt file**

1979 23 23 2 43 24 25 26 26 26 26 25 26 25

1980 26 27 28 28 28 30 31 31 31 30 30 30 29

1981 31 32 32 32 33 34 35 36 36 34 34 34 34

1984 39 38 39 39 39 41 42 43 40 39 38 38 40

1985 38 39 39 39 39 41 41 41 00 40 39 39 45

### Code:

package hadoop;

import java.util.\*;

import java.io.IOException;

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

import org.apache.hadoop.util.\*;

public class ProcessUnits {

//Mapper class

public static class E\_EMapper extends MapReduceBase implements

Mapper<LongWritable ,/\*Input key Type \*/

Text, /\*Input value Type\*/

Text, /\*Output key Type\*/

IntWritable> /\*Output value Type\*/

{

//Map function

public void map(LongWritable key, Text value,

OutputCollector<Text, IntWritable> output,

Reporter reporter) throws IOException {

String line = value.toString();

String lasttoken = null;

StringTokenizer s = new StringTokenizer(line,"\t");

String year = s.nextToken();

while(s.hasMoreTokens()) {

lasttoken = s.nextToken();

}

int avgprice = Integer.parseInt(lasttoken);

output.collect(new Text(year), new IntWritable(avgprice));

}

}

//Reducer class

public static class E\_EReduce extends MapReduceBase implements Reducer< Text, IntWritable, Text, IntWritable > {

//Reduce function

public void reduce( Text key, Iterator <IntWritable> values,

OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

int maxavg = 30;

int val = Integer.MIN\_VALUE;

while (values.hasNext()) {

if((val = values.next().get())>maxavg) {

output.collect(key, new IntWritable(val));

}

}

}

}

//Main function

public static void main(String args[])throws Exception {

JobConf conf = new JobConf(ProcessUnits.class);

conf.setJobName("max\_eletricityunits");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(E\_EMapper.class);

conf.setCombinerClass(E\_EReduce.class);

conf.setReducerClass(E\_EReduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

JobClient.runJob(conf);

}

}

The following command is used to run the Eleunit\_max application by taking the input files from the input directory.

$HADOOP\_HOME/bin/hadoop jar units.jar hadoop.ProcessUnits input\_dir output\_dir

Wait for a while until the file is executed. After execution, as shown below, the output will contain the number of input splits, the number of Map tasks, the number of reducer tasks, etc.

INFO mapreduce.Job: Job job\_1414748220717\_0002

completed successfully

14/10/31 06:02:52

INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read = 61

FILE: Number of bytes written = 279400

FILE: Number of read operations = 0

FILE: Number of large read operations = 0

FILE: Number of write operations = 0

HDFS: Number of bytes read = 546

HDFS: Number of bytes written = 40

HDFS: Number of read operations = 9

HDFS: Number of large read operations = 0

HDFS: Number of write operations = 2 Job Counters

Launched map tasks = 2

Launched reduce tasks = 1

Data-local map tasks = 2

Total time spent by all maps in occupied slots (ms) = 146137

Total time spent by all reduces in occupied slots (ms) = 441

Total time spent by all map tasks (ms) = 14613

Total time spent by all reduce tasks (ms) = 44120

Total vcore-seconds taken by all map tasks = 146137

Total vcore-seconds taken by all reduce tasks = 44120

Total megabyte-seconds taken by all map tasks = 149644288

Total megabyte-seconds taken by all reduce tasks = 45178880

Map-Reduce Framework

Map input records = 5

Map output records = 5

Map output bytes = 45

Map output materialized bytes = 67

Input split bytes = 208

Combine input records = 5

Combine output records = 5

Reduce input groups = 5

Reduce shuffle bytes = 6

Reduce input records = 5

Reduce output records = 5

Spilled Records = 10

Shuffled Maps = 2

Failed Shuffles = 0

Merged Map outputs = 2

GC time elapsed (ms) = 948

CPU time spent (ms) = 5160

Physical memory (bytes) snapshot = 47749120

Virtual memory (bytes) snapshot = 2899349504

Total committed heap usage (bytes) = 277684224

File Output Format Counters

Bytes Written = 40

The following command is used to see the output in **Part-00000**file. This file is generated by HDFS.

$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00000

**OUTPUT:**

1981 34

1984 40

1985 45

**PRACTICAL 3**

**Aim:**

Plot a live graph of a sentimental analysis in twitter

## CODE:

import dash

from dash.dependencies import Output, Event

import dash\_core\_components as dcc

import dash\_html\_components as html

import plotly

import random

import plotly.graph\_objs as go

from collections import deque

X = deque(maxlen=20)

X.append(1)

Y = deque(maxlen=20)

Y.append(1)

app = dash.Dash(\_\_name\_\_)

app.layout = html.Div(

[

dcc.Graph(id='live-graph', animate=True),

dcc.Interval(

id='graph-update',

interval=1\*1000

),

]

)

@app.callback(Output('live-graph', 'figure'),

events=[Event('graph-update', 'interval')])

def update\_graph\_scatter():

X.append(X[-1]+1)

Y.append(Y[-1]+Y[-1]\*random.uniform(-0.1,0.1))

data = plotly.graph\_objs.Scatter(

x=list(X),

y=list(Y),

name='Scatter',

mode= 'lines+markers'

)

return {'data': [data],'layout' : go.Layout(xaxis=dict(range=[min(X),max(X)]),

yaxis=dict(range=[min(Y),max(Y)]),)}

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

import sqlite3

import pandas as pd

conn = sqlite3.connect('twitter.db')

c = conn.cursor()

df = pd.read\_sql("SELECT \* FROM sentiment WHERE tweet LIKE '%olympic%' ORDER BY unix DESC LIMIT 1000", conn)

df.sort\_values('unix', inplace=True)

df['sentiment\_smoothed'] = df['sentiment'].rolling(int(len(df)/5)).mean()

df.dropna(inplace=True)

X = df.unix.values[-100:]

Y = df.sentiment\_smoothed.values[-100:]

import dash

from dash.dependencies import Output, Event

import dash\_core\_components as dcc

import dash\_html\_components as html

import plotly

import random

import plotly.graph\_objs as go

from collections import deque

import sqlite3

import pandas as pd

#popular topics: google, olympics, trump, gun, usa

app = dash.Dash(\_\_name\_\_)

app.layout = html.Div(

[ html.H2('Live Twitter Sentiment'),

dcc.Graph(id='live-graph', animate=True),

dcc.Interval(

id='graph-update',

interval=1\*1000

),

]

)

@app.callback(Output('live-graph', 'figure'),

events=[Event('graph-update', 'interval')])

def update\_graph\_scatter():

try:

conn = sqlite3.connect('twitter.db')

c = conn.cursor()

df = pd.read\_sql("SELECT \* FROM sentiment WHERE tweet LIKE '%olympic%' ORDER BY unix DESC LIMIT 1000", conn)

df.sort\_values('unix', inplace=True)

df['sentiment\_smoothed'] = df['sentiment'].rolling(int(len(df)/5)).mean()

df.dropna(inplace=True)

X = df.unix.values[-100:]

Y = df.sentiment\_smoothed.values[-100:]

data = plotly.graph\_objs.Scatter(

x=X,

y=Y,

name='Scatter',

mode= 'lines+markers'

)

return {'data': [data],'layout' : go.Layout(xaxis=dict(range=[min(X),max(X)]),

yaxis=dict(range=[min(Y),max(Y)]),)}

except Exception as e:

with open('errors.txt','a') as f:

f.write(str(e))

f.write('\n')

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(debug=True)

from tweepy import Stream

from tweepy import OAuthHandler

from tweepy.streaming import StreamListener

import json

import sqlite3

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

from unidecode import unidecode

import time

analyzer = SentimentIntensityAnalyzer()

conn = sqlite3.connect('twitter.db')

c = conn.cursor()

def create\_table():

try:

c.execute("CREATE TABLE IF NOT EXISTS sentiment(unix REAL, tweet TEXT, sentiment REAL)")

c.execute("CREATE INDEX fast\_unix ON sentiment(unix)")

c.execute("CREATE INDEX fast\_tweet ON sentiment(tweet)")

c.execute("CREATE INDEX fast\_sentiment ON sentiment(sentiment)")

conn.commit()

except Exception as e:

print(str(e))

create\_table()

#consumer key, consumer secret, access token, access secret.

ckey=""

csecret=""

atoken=""

asecret=""

class listener(StreamListener):

def on\_data(self, data):

try:

data = json.loads(data)

tweet = unidecode(data['text'])

time\_ms = data['timestamp\_ms']

vs = analyzer.polarity\_scores(tweet)

sentiment = vs['compound']

print(time\_ms, tweet, sentiment)

c.execute("INSERT INTO sentiment (unix, tweet, sentiment) VALUES (?, ?, ?)",

(time\_ms, tweet, sentiment))

conn.commit()

except KeyError as e:

print(str(e))

return(True)

def on\_error(self, status):

print(status)

while True:

try:

auth = OAuthHandler(ckey, csecret)

auth.set\_access\_token(atoken, asecret)

twitterStream = Stream(auth, listener())

twitterStream.filter(track=["a","e","i","o","u"])

except Exception as e:

print(str(e))

time.sleep(5)

## OUTPUT:

## Chart, line chart Description automatically generated

## Chart, line chart Description automatically generated