빅데이터 분석

- Chapter02

데이터는 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz' 에서 다운을 받았다.

Index of /databases/kddcup99

- Parent Directory
- corrected.gz
- kddcup.data.gz
- kddcup.data_10_percent.gz
- kddcup.names
- kddcup.newtestdata_10_percent_unlabeled.gz
- kddcup.testdata.unlabeled.gz
- kddcup.testdata.unlabeled_10_percent.gz
- kddcup99.html
- task.html
- training_attack_types
- typo-correction.txt

Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips SVN/1.7.14 Server at kdd.ics.uci.edu Port 80

```
[1] import urllib.request
[2] url = 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz'
[3] localfile = '/tmp/kddcup.data_10_percent.gz'
[4] f = urllib.request.urlretrieve(url,localfile)
```

[1] URL(Uniform Resource Locator)를 가져 오기 위한 파이썬 모듈이다.urlopen 함수의 형태로 매우 간단한 인터페이스를 제공한다, 다양한 프로토콜을 사용해 URL을 가져올 수 있다.

[4] urllib.request.urlretrieve 함수를 통해 바로 파일에 자료를 입력할 수 있다.

=> URL에 있는 데이터를 localfile 경로 안에 저장한다.

```
[1] raw_data = sc.textFile("file:///tmp/kddcup.data_10_percent.gz")
[2] raw_data.take(2)
```

```
['0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00
,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.',

'0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0
.00,19,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,normal.']
```

[1] “file://“ : 로컬 네트워크의 파일을 참조하는 프로토콜

[1] sc.textFile : kddcup.data_10_percent.gz을 RDD로 정의한다.
[2] .take(N) : 처음 n개를 return 해준다. head랑 비슷한 듯?

```
raw_data
```

```
[out] : file:///tmp/kddcup.data_10_percent.gz MapPartitionsRDD[5] at textFile at NativeMethodAccessorImpl.java:0
```

```
a = range(100)
print(list(a))
```

‘

```
[1] list_rdd=sc.parallelize(list(a))
list_rdd
```

```
[out] :  ParallelCollectionRDD[7] at parallelize at PythonRDD.scala:195
```

[1] sc.parallelize : RDD 생성, 뒤에 따로 숫자가 없으므로 데이터를 파티션으로 나누지 않고 넣는다. 만약 sc.parallelize(data,8) 이런식이면 해당 데이터를 8개의 파티션으로 나눈다는 의미이다.

```
list_rdd.count()
```

```
[out] : 100
```

```
list_rdd.take(10)
```

```
[out] : [0,1,2,3,4,5,6,7,8,9]
```

```
len(a)
```

```
[out] : 100
```

```
[1] list_rdd.reduce(lambda a,b : a+b)
```

```
[out] : 4950
```

[1] lambda : 런타임에서 이름을 할당 받을 필요가 없는 한 줄 짜리 익명 함수를 만들고 싶은 경우 사용한다. 위는 숫자를 받아와 덧셈 연산을 한다. 따라서 0부터 100까지 전부 다 더해주면 된다.
이렇게 계산된다.
https://backtobazics.com/big-data/spark/apache-spark-reduce-example/

```
[1] contains_normal = raw_data.filter(lambda line: "normal." in line )
contains_normal.count()
```

```
[out] : 97278
```

[1] filter() : 주어진 조건에 해당하는 데이터를 선별한다. filter 메소드는 이미 존재하는 RDD

를 변경하는 것이 아니라 완전히 새로운 RDD에 대한 포인터를 리턴한다. transformation 함수라고 한다.
관련 함수에 대한 설명은 여기서 보면 될 듯
https://whereami80.tistory.com/102?category=649606

```
split_file = raw_data.map(lambda line: line.split(','))
for i in split_file.take(2):
    print(i)
```

```
[out] : ['0', 'tcp', 'http', 'SF', '181', '5450', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '8', '8', '0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '9', '9', '1.00', '0.00', '0.11', '0.00', '0.00', '0.00', '0.00', '0.00', 'normal.']
['0', 'tcp', 'http', 'SF', '239', '486', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '8', '8', '0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '19', '19', '1.00', '0.00', '0.05', '0.00', '0.00', '0.00', '0.00', '0.00', 'normal.']
```

```
raw_data.collect()
```

```

```

- Chapter3

```
from time import time
from IPython.core.magics.execution import _format_time as format_delta

start = time()
raw_data = sc.textFile("file:///tmp/kddcup.data_10_percent.gz")
stop_time = time()
print('time : {}'.format(format_delta(stop_time-start_time))
```

```
[out] : time : 88.8ms
```

1) 데이터 전체에 대한 시간 측정

```
[1] contains_normal = raw_data.map(lambda x:x.split(",")).filter(lambda x:"normal" in x)
[2] t0 = time()
[3] num_sampled = contains_normal.count()
[4] duration = time() - t0
[5] duration
```

```
[out] : 3.12 s
```

[1] map() : 데이터를 가공한다. 반환 type이 같지 않아도 된다. map 함수의 경우 리스트 안에 또 리스트가 있는 구조를 보존하고 처리한다. 따라서 파일의 각 줄마다 리스트를 만든다. map 함수에 대한 예시이다.
https://parkaparka.tistory.com/15

2) Sampled 에 대한 시간 측정

```
[1] sampled = raw_data.sample(False,0.1,42)
[2] contains_normal_sample = sampled.map(lambda x : x.split(',')).filter( lambda x : "normal" in x)
[3] t0 = time()
[4] num_sampled = contains_normal_sample.count()
[5] duration = time() - t0
[6] duration
```

```
[out] : time : 1.21 s
```

[1] .sample : raw_data에서 sample을 뽑아 낸다. 첫 번째 인자는 True면 복원추출, False면 비복원추출을 실행한다. 복원 추출이란 한 번 뽑은 것을 다시 뽑을 수 있게 하는 방법이다. 두 번째 인자는 데이터의 몇퍼센트를 뽑아내겠다는 의미인데, 위 예제에서는 10%를 뽑아내는 것을 알 수 있다. 세 번째 인자로 시드 변수를 지정할 수 있다. 위 예제에서는 random seed로 42를 선택했다.

duration(기간)을 비교한 결과 전체 데이터를 다 넣고 측정한 시간보다 일부만 뽑아서 측정한 시간이 훨씬 짧음을 알 수 있다.

```
[1] data_in_memory = raw_data.takeSample(False, 10, 42) # type : list
[2] contains_normal_py =[line.split(',') for line in data_in_memory if "normal" in line]
[3] len(contains_normal_py)
```

```
[out] : 1998
```

[1] .takeSample : 모든 데이터가 드라이버의 메모리에 로드되기 때문에 결과 배열이 작을 것으로 예상되는 경우 사용한다. 두 번째 인자를 지정하여 몇 개를 추출할 것인지 정할 수 있다.

원래는 1000개의 데이터 포인트의 샘플을 가져왔는데 이로 인해 시스템이 충돌됨을 확인하였다. 그래서 10개의 데이터 포인트를 가지고 normal을 포함하는지 확인해 보았다.
Spark를 사용하면 큰 데이터 셋을 병렬화하고 병렬 방식으로 작업할 수 있다. 즉, 적은메모리와 적은 시간으로 더 많은 작업을 수행할 수 있다.

```
[1] normal_sample = sampled.filter(lambda line:"normal." in line)
[2] non_normal_sample = sampled.subtract(normal_sample) # minus
print(sample.count())
print(normal_sample.count())
print(non_normal_sample.count())
```

```
개수들이 각각 나오겠지 !! 파일 저장 잘못해서 output 모름
```

[2] .substract는 minus를 나타낸다. 따라서 sampled에서 normal_sample 데이터를 뺀 나머지 데이터가 non_normal_Sample이 된다.

Cartesian : Return the Cartesian product of this RDD and another one, that is, the RDD of all pairs of elements (a, b) where a is in self and b is in other.

```
[1] feature_1 = sampled.map(lambda line:line.split(",")).map(lambda features:features[1]).distinct()

[2] feature_2 = sampled.map(lambda line:line.split(",")).map(lambda features:features[2]).distinct()

[3] f1 = feature_1.collect()
[4] f2 = feature_2.collect()
print(f1)
```

```
print(f2)
```

```
[out] : ['tcp', 'icmp', 'udp']
['http', 'smtp', 'auth', 'ecr_i', 'finger', 'ftp', 'domain_u', 'ntp_u', 'eco_i', 'private', 'ftp_data', 'telnet', 'pop_3', 'mtp', 'link', 'gopher', 'other', 'IRC', 'klogin', 'echo', 'time', 'remote_job', 'hostnames', 'uucp_path', 'nntp', 'http_443', 'efs', 'uucp', 'sql_net', 'daytime', 'rje', 'csnet_ns', 'sunrpc', 'bgp', 'vmnet', 'nnsp', 'whois', 'domain', 'printer', 'kshell', 'iso_tsap', 'name', 'supdup', 'pop_2', 'ldap', 'login', 'netbios_ns', 'imap4', 'Z39_50', 'discard', 'systat', 'exec', 'netstat', 'netbios_dgm', 'urh_i', 'urp_i', 'courier', 'ctf', 'shell', 'netbios_ssn', 'ssh', 'X11']
```

```
#f2 has a lot more values, and we can use the cartesian function
# to clooect all the combinations between f1 and f2 as follows :
len(feature_1.cartesian(feature_2).collect())
```

```
```

<cartesian>

scala> val rdd1 = sc.parallelize(List("Spark","Scala"))
rdd1: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[50] at parallelize at <console>:24
scala> val rdd2 = sc.parallelize(List("Akka","Scala"))
rdd2: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[51] at parallelize at <console>:24

scala> rdd1.cartesian(rdd2).collect()
res13: Array[(String, String)] = Array((Spark,Akka), (Spark,Scala), (Scala,Akka), (Scala,Scala))
출처: https://knight76.tistory.com/entry/spark-집합-함수-union-intersection-cartesian-subtract-join-cogroup-예제 [김용환 블로그(2004-2020)]

-Chapter4 : Aggregating and Summarizing Data into Useful Reports

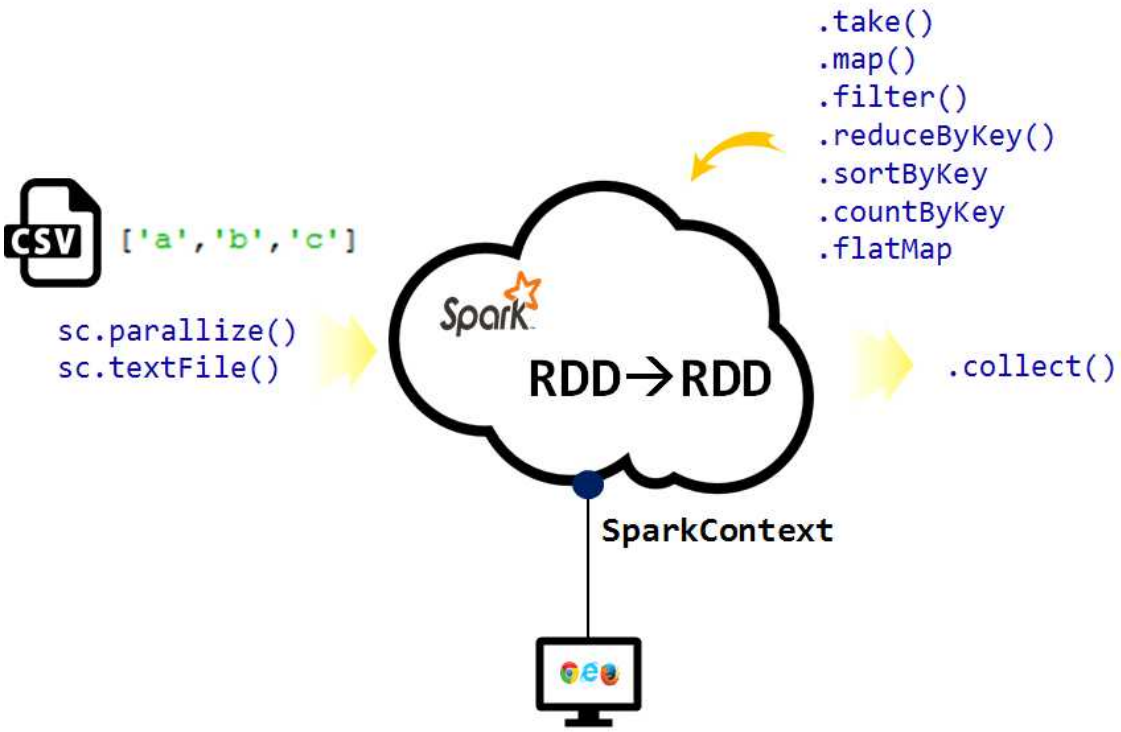
1) How do we calculate averages, map, reduce?

```
# Create three tuple
[1] rdd = sc.parallelize(['b','a','c'])
[2] sorted(rdd.map(lambda x : (x,1)).collect())
```

```
[out] : [('a',1),('b',2),('c',3)]
```

[2] sorted가 있기 때문에 정렬해서 output에 나와야 한다.collect() 함수로 해당 데이터의 모든 row를 반환해서 적어주면 된다.

아래 사진은 RDD 자료변환의 개요이다.
여러 파일들을 스파크로 불러들인 후, 함수들을 활용해 원하는 형태로 데이터를 가공시킨다. 그 후, .collect() 함수로 스파크 RDD를 뽑아낸다.



https://statkclee.github.io/bigdata/bigdata-pyspark-data-transformation.html

```
[1] raw_data = sc.textFile("file:///tmp/kddcup.data_10_percent.gz")
[2] csv = raw_data.map(lambda x:x.split(","))
[3] normal_data = csv.filter(lambda x:x[41]=="normal.")
[4] duration = normal_data.map(lambda x:int(x[0])) # new RDDs
[5] total_duration = duration.reduce(lambda x,y:x+y)
[6] total_duration

# to divide total_duration by the counts of the data as follows:
[7] total_duration/(normal_data.count())
```

```
[out] : 21075991, 216.6573
```

2) Faster average computations with aggregate

.aggregate(zeroValue, seq0p, com0p)

zeroValue : Aggregate the elements of each partition, and then the results for all the partitions, using a given combine functions and a neutral “zero value.”
반환하려는 유형의 초기값을 제공한다.
seq0p : The functions op(t1, t2) is allowed to modify t1 and return it as its result value to avoid object allocation; however, it should not modify t2.
t2 타입을 연산 결과 t1으로 변경 및 연산을 수행해야 하는 함수이다. 이 함수는 전역적인 작업이 아니라 각 노드 로컬 파티션에서 수행된다.
com0p : The first function (seqOp) can return a different result type, U, than the type of this RDD. Thus, we need one operation for merging a T into an U and one operation for merging two U
각 파티션에서 seq0o 작업이 끝난 후 합치는 과정이다.
즉, 파티션 단위의 연산을 한 후 각 연산의 결과를 합친다.

예시)

```
>>> seq0p = (lambda x, y : (x[0] + y, x[1] + 1))
>>> com0p = (lambda x, y : (x[0] + y[0], x[1] + y[1]))
>>> sc.parallelize([1,2,3,4]).aggregate((0,0), seq0p, com0p)
(10,4)
>>> sc.parallelize([]).aggregate((0,0), seq0p, com0p)
(0,0)
```

```
duration_count = duration.aggregate(
    (0,0),
     lambda db, new_value: (db[0]+new_value, db[1]+1),
     lambda db1, db2: (db1[0]+db2[0], db1[1]+db2[1])
```

```
)
print("average duration : {}".format(duration_count[0]/duration_count[1]))
```

```
[out] : average duration: 216.65732231336992
```

3) Pivot tabling with key-value paired data points

```
kv = csv.map(lambda x:(x[41],x))
print(kv.take(1))
```

```
[out] : [('normal.', ['0', 'tcp', 'http', 'SF', '181', '5450', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '8', '8', '0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '9', '9', '1.00', '0.00', '0.11', '0.00', '0.00', '0.00', '0.00', '0.00', 'normal.'])]
```

튜플 리스트를 스파크에 넣어서 RDD로 변환시킨 후, reduceByKey() 함수를 사용해 키 값을 기준으로 value에 대한 연산 작업을 수월하게 진행한다.

```
kv_duration = csv.map(lambda x: (x[41],float(x[0]))).reduceByKey(lambda x,y: x+y)
kv_duration.collect()
```

```
[out] : [('normal.', 21075991.0),
 ('buffer_overflow.', 2751.0),
 ('loadmodule.', 326.0),
 ('perl.', 124.0),
 ('neptune.', 0.0),
 ('smurf.', 0.0),
 ('guess_passwd.', 144.0),
 ('pod.', 0.0),
 ('teardrop.', 0.0),
 ('portsweep.', 1991911.0),
 ('ipsweep.', 43.0),
 ('land.', 0.0),
 ('ftp_write.', 259.0),
 ('back.', 284.0),
 ('imap.', 72.0),
 ('satan.', 64.0),
 ('phf.', 18.0),
 ('nmap.', 0.0),
 ('multihop.', 1288.0),
```

```
 ('warezmaster.', 301.0),
 ('warezclient.', 627563.0),
 ('spy.', 636.0),
 ('rootkit.', 1008.0)]
```

잘 정렬된 모습이 보인다.

```
kv.countByKey()
```

```
[out] : defaultdict(int,
                    {'normal.': 97278,
                     'buffer_overflow.': 30,
                     'loadmodule.': 9,
                     'perl.': 3,
                     'neptune.': 107201,
                     'smurf.': 280790,
                     'guess_passwd.': 53,
                     'pod.': 264,
                     'teardrop.': 979,
                     'portsweep.': 1040,
                     'ipsweep.': 1247,
                     'land.': 21,
                     'ftp_write.': 8,
                     'back.': 2203,
                     'imap.': 12,
                     'satan.': 1589,
                     'phf.': 4,
                     'nmap.': 231,
                     'multihop.': 7,
                     'warezmaster.': 20,
                     'warezclient.': 1020,
                     'spy.': 2,
                     'rootkit.': 10})
```

key에 대한 unique set의 count가 저장된다.

별개로 sc 에러 나면
>>> sc.stop() 해주고 다시 실행하면 된다.

https://12bme.tistory.com/434 <- rdd 설명
정리하자면,

Spark의 작동원리는 크게 두가지로 나뉜다.
1) transformation : 수행하고 싶은 operation으로 RDD transformation
(read -> parser -> selection)
- RDD는 병렬로 동시에 실행이 된다.
- textFile() 등등으로 raw data를 RDD로 변한한다.
- map
2) action : 선언된 RDD 결과값을 출력한다.
-collect
-countByKey()
-take() 등등