

PySpark Grundlagen

PySpark ist eine Python-API für Apache Spark, ein leistungsstarkes open-source Framework für Cluster und Datenanalyse.

SparkContext initialisieren

```
from pyspark import SparkContext
sc = SparkContext(master = 'local[2]')
```

SparkContext Informationen

sc.version	Version
sc.pythonVer	Python Version
sc.master	Master URL
str(sc.sparkHome)	Pfad zu Spark auf den Nodes
str(sc.sparkUser())	Benutzername
sc.appName	Name der App
sc.applicationId	ID der App
sc.defaultParallelism	Parallelisierung
sc.defaultMinPartitions	Minimale Anzahl Partitionen für RDDs

Konfiguration

```
from pyspark import SparkConf, SparkContext
```

```
conf = (SparkConf()
        .setMaster("local")
        .setAppName("My app")
        .set("spark.executor.memory", "1g"))
```

```
sc = SparkContext(conf = conf)
```

PySpark Shell

PySpark hat eine eigene Shell, in der ein SparkContext **sc** bereits erzeugt ist:

```
./bin/spark-shell -master local[2]
./bin/pyspark -master local[4] -py-files code.py
```

Dabei wird der verwendete Master mit **-master** und Python-Scripte, EGG-, oder ZIP-Dateien als durch Kommas getrennte Dateinamen mit **-py-files** übergeben.

Daten laden

```
Externe Daten:
textFile = sc.textFile("/mein/ordner/*.txt")
textFile2 = sc.wholeTextFiles("/mein/ordner/")
```

```
Parallelisierte Verbindungen:
rdda = sc.parallelize([('a',7),('a',2),('b',2)])
rddb = sc.parallelize([('a',2),('d',1),('b',1)])
rddc = sc.parallelize(range(100))
```

```
Json:
df = spark.read.json('pfad/datei.json')
```

rdda-Informationen auslesen

Grundlagen

rdda.getNumPartitions()	Anzahl Partitionen
rdda.count()	Anzahl rdda Instanzen
3	
rdda.countByKey()	Anzahl Instanzen nach Schlüssel
defaultdict(<type 'int'>, 'a':2, 'b':1)	
rdda.countByValue()	Anzahl Instanzen nach Wert
defaultdict(<type 'int'>, ('b',2):1, ('a',2):1, ('a',7):1)	
rdda.collectAsMap()	Instanzen als Dictionary
'a': 2, 'b': 2	
rddc.sum()	Summe der rdda-Elemente
4950	
sc.parallelize([]).isEmpty()	Ist das rdda leer?
True	

Gesamtwerte

rddc.max()	Maximum
99	
rddc.min()	Minimum
0	
rddc.mean()	Mittelwert
49.5	
rddc.stdev()	Standardabweichung
28.866070047722118	
rddc.variance()	Varianz
833.25	
rddc.histogram(3)	Histogramm mit 3 Bins
([0,33,66,99],[33,33,34])	
rddc.stats()	Alle obigen Gesamtwerte

Iterationen

Eine Funktion auf alle RDDs anwenden:

Funktion	Ausgabe
def g(x): print(x)	('a', 7)
rdda.foreach(g)	('b', 2)
	('a', 2)

Daten manipulieren

Funktionen anwenden

rdda.map(lambda x: x+(x[1],x[0]))	Funktion auf alle Elemente anwenden
[('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]	
rdd5 = rdda.flatMap(lambda x: x+(x[1],x[0]))	Auf alle Elemente anwenden und verflachen
rdd5.collect()	
[('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]	

Daten auswählen

rdda.collect()	Liste aller Elemente
[('a', 7), ('a', 2), ('b', 2)]	
rdda.first()	Erstes Element
('a', 7)	
rdda.take(n)	Erste n Elemente
[('a', 7), ('a', 2)]	
rdda.take(2)	Beste zwei Elemente
[('b', 2), ('a', 7)]	
rddc.sample(False, 0.15, 81).collect()	Randomisierte Teilmenge
[3,4,27,31,40,41,42,43,60,76,79,80,86,97]	
rdda.filter(lambda x: "a" in x).collect()	Filterfunktion anwenden
[('a',7),('a',2)]	
rdd5.distinct().collect()	Verschiedene Werte filtern
[('a',2),('b',7)]	

Mathematische Operationen

rdda.subtract(rddb).collect()	Jeder Wert von rdda der nicht in rddb ist
[('b',2),('a',7)]	
rdda.subtractByKey(rddb).collect()	(Schlüssel, Wert) für Schlüssel exklusiv in rdda
[('d', 1)]	
rdda.cartesian(rddb).collect()	Kartesisches Produkt von rdda und rddb

Weitere Themen

DataFrame aus RDD

```
df = rdd.map(lambda x: Row(**f(x))).toDF()
df = sqlContext.createDataFrame(rdd, schema)
```

Fehlende Werte

df.na.fill()	Alle null -Werte ersetzen
df.na.drop()	Alle Zeilen mit null -Werten löschen

Daten zusammenfügen

```
left.join(right,key, how='*')
```

* = left, right, inner, full

Filtern

```
df.filter(df["alter"]>24)
```

Nur Einträge mit **alter** größer 24 behalten

Sortieren

rddb.sortBy(lambda x: x[1]).collect()	Sortierfunktion
[('d',1),('b',1),('a',2)]	
rddb.sortByKey()	Nach Schlüssel
[('a',2),('b',1),('d',1)]	

Repartitionieren

rdda.repartition(4)	Neue rdda mit 4 Partitionen
rdda.coalesce(1)	Anzahl Partitionen auf 1 reduzieren

Speichern

```
rdda.saveAsTextFile("rdda.txt")
rdda.saveAsHadoopFile("hdfs://nodehost/parent/child",
'org.apache.hadoop.mapred.TextOutputFormat'
)
```

SparkContext beenden

```
sc.stop()
```

Ausführen

```
./bin/spark-submit \
examples/src/main/python/pi.py
```


DataFrames zusammenfügen

Verändernde Joins

A
X1 X2
a 1
b 2
c 3

B
X1 X3
a T
b F
d T

X1 X2 X3
a 1 T
b 2 F
c 3 T

X1 X2 X3
a 1 T
b 2 F
c null T

X1 X2 X3
a 1 T
b 2 F

X1 X2 X3
a 1 T
b 2 F
d null T

Zeilenweise von **B** nach **A**
A.join(B,'X1',how='left')
.orderBy('X1',
ascending=True).show()

Zeilenweise von **A** nach **B**
A.join(B,'X1',how='right')
.orderBy('X1',
ascending=True).show()

Nur gemeinsame Zeilen
A.join(B,'X1',how='inner')
.orderBy('X1',
ascending=True).show()

Alle Zeilen
A.join(B,'X1',how='full')
.orderBy('X1',
ascending=True).show()

Joins filtern

X1 X2
a 1
b 2

In **B** enthaltene **A** Zeilen

A.join(B,'X1',how='left_semi')
.orderBy('X1',
ascending=True).show()

In **B** nicht enthaltene **A** Zeilen
A.join(B,'X1',how='left_anti')
.orderBy('X1',
ascending=True).show()

X1 X2
C 3

DataFrame Operationen

Y
X1 X2
a 1
b 2
c 3

Z
X1 X2
b 2
c 3
d 4

X1 X2
b 2
c 3

X1 X2
a 1
b 2
c 3
d 4

X1 X2
b 2
c 3

X1 X2
a 1

Zeilen in **Y** und **Z**

Y.intersect(Z).show()

Zeilen in **Y** und/oder **Z**

Y.union(Z).dropDuplicates()
.orderBy('X1',
ascending=True).show()

Zeilen nur in **Y**

Y.subtract(Z).show()

Daten bearbeiten

Verbinden

Z als neue Zeilen anbinden

Y.union(Z) .orderBy('X1',
ascending=True).show()

X1 X2
a 1
b 2
c 3
b 2
c 3
d 4

Z als neue Spalten anbinden

zipDataFrames(Y,Z).show()

X1 X2 X1 X2
a 1 b 2
b 2 c 3
c 3 d 4

Aufteilen

Zellen aufteilen

df.select("key", df.value[0], df.value[1], df.value[2]).show()

key value
a [1,2,3]
b [2,3,4]

Spalten in Zeilen aufteilen

df.select("key",F.split("values", ",").alias("values"),
F.posexplode(F.split("values",",")) .alias("pos", "val")).drop("val")
.select("key",F.expr("values[pos]").alias("val")).show()

key value
a 1,2,3
b 2,3,4

Spalten in Zeilen überführen

def to_long(df, by):
cols, dtypes = zip(*(c,t) for (c, t) in df.dtypes if c not in by))
kvs = explode(array([
struct(lit(c).alias("key"), col(c).alias("val")) for c in cols
])).alias("kvs")
return df.select(by + [kvs]).select(by + ["kvs.key", "kvs.val"])

A val1 val2 val3
a 1 2 3
b 4 5 6

Pivotieren

key value
a 1
a 2
a 3
a 1
b 1
b 2

df.groupBy(['key']) .pivot('col1').sum('col1').show()

key 1 2 3
b 1 2 null
a 4 2 3

Gruppieren

A B C
m 1 4
m 2 5
n 3 7
n 4 8

df.groupBy(['A'])
.agg(F.min('B').alias('min_b'),
F.max('B').alias('max_b'),
F.avg('C').alias('avg_c')).show()

A min_b max_b avg_c
m 1 2 4.5
n 3 4 7.5

A min_b max_b list_c
m 1 2 [4.2,4.5,4.75]
n 3 4 [7.2,7.5,7.75]

def quant_pd(val_list):
quant = np.round(np.percentile(val_list,[20,50,75]),2)
return list(map(float,quant))
Fn = F.udf(quant_pd,ArrayType(FloatType()))

df.groupBy(['A']).agg(F.min('B').alias('min_b'),
F.max('B').alias('max_b'),
Fn(F.collect_list(col('C'))).alias('list_c'))

Daten bearbeiten

Neue Variablen erzeugen

df.withColumn('new',1/df.col)

df.withColumn('new',F.log(df.col))
df.withColumn("new", Fn('col'))

Teilmengen (Zeilen)

df.na.drop()

Zeilen mit **null** auslassen

df.filter() Zeilen filtern

df.distinct() Verschiedene Zeilen extrahieren

df.sample() Zeilen sampeln

Teilmengen (Spalten)

df.select()

Bilde mit dem Ausdruck ein neues Dataframe

Daten zusammenfassen

df.describe()

Berechnet einfache Statistiken


Correlation.corr(df) Korrelationskoeffizientenmatrix

df.count() Zählt die Zeilen

Variablen umbenennen

df.withColumnRenamed()

Spaltennamen ändern



DATAMICS

machine intelligence consulting services