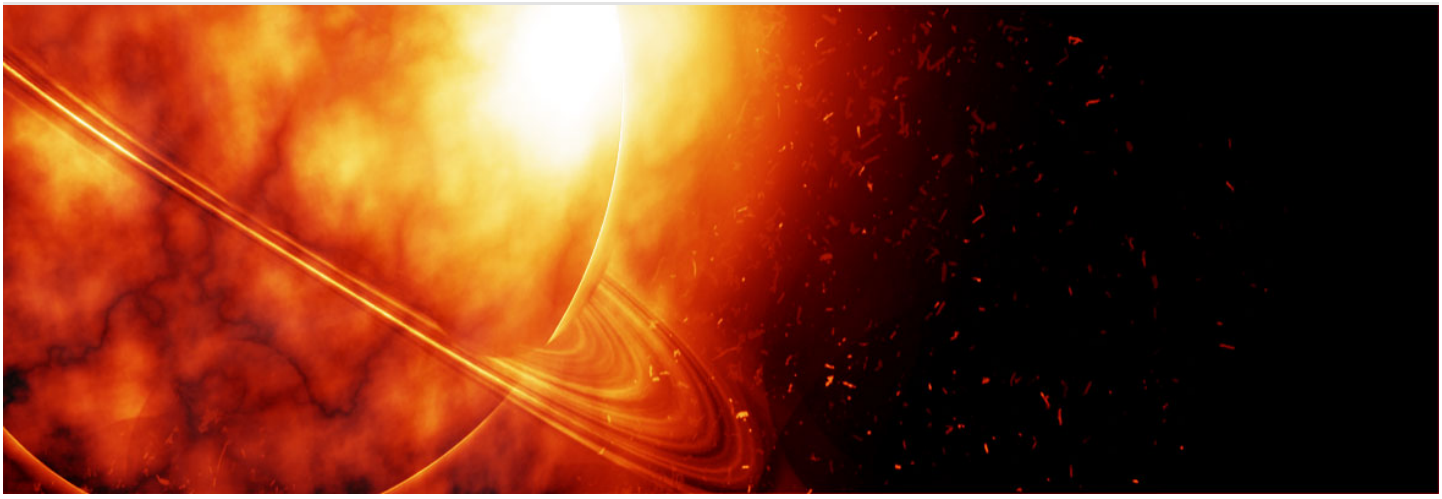
MungingData

Piles of precious data



Advanced String Matching with Spark's rlike Method

mrpowers April 6, 2018 1

The Spark `rlike` method allows you to write powerful string matching algorithms with regular expressions (regexp).

This blog post will outline tactics to detect strings that match multiple different patterns and how to abstract these regular expression patterns to CSV files.

Writing Beautiful Spark Code is the best way to learn how to use regular expressions when working with Spark StringType columns.

Substring matching

Let's create a DataFrame and use `rlike` to identify all strings that contain the substring `"cat"`.

```
val df = List(
  ("the cat and the hat"),
  ("i love your cat"),
  ("dogs are cute"),
  ("pizza please")
).toDF("phrase")

df
  .withColumn(
    "contains_cat",
    col("phrase").rlike("cat")
  )
  .show(truncate = false)
```

```
+------------------+------------+
|phrase            |contains_cat|
+------------------+------------+
|the cat and the hat|true        |
|i love your cat    |true        |
|dogs are cute      |false       |
|pizza please       |false       |
+------------------+------------+
```

There is nothing special about this example and if you're only looking to match a single substring, it's better to use `contains` than `rlike`.

Let's rework this code to detect all strings that contain the substrings `"cat"` or `"dog"`.

```
df
  .withColumn(
    "contains_cat_or_dog",
    col("phrase").rlike("cat|dog")
  )
  .show(truncate = false)
```

```
+------------------+------------+
|phrase            |contains_cat|
+------------------+------------+
|the cat and the hat|true        |
|i love your cat    |true        |
|dogs are cute      |true        |
|pizza please       |false       |
+------------------+------------+
```

We can refactor this code by storing the animals in a list and concatenating them as a pipe delimited string for the `rlike` method.

```
val animals = List("cat", "dog")

df
  .withColumn(
    "contains_cat_or_dog",
    col("phrase").rlike(animals.mkString("|"))
  )
  .show(truncate = false)
```

```
+------------------+------------------+
|phrase            |contains_cat_or_dog|
+------------------+------------------+
|the cat and the hat|true              |
|i love your cat    |true              |
|dogs are cute      |true              |
|pizza please       |false             |
+------------------+------------------+
```

Matching strings that start with or end with substrings

Let's create a new DataFrame and match all strings that begin with the substring `"i like"` or `"i want"`.

```
val df = List(
  ("i like tacos"),
  ("i want love"),
  ("pie is what i like"),
  ("pizza pizza"),
  ("you like pie")
).toDF("phrase")

df
  .withColumn(
    "starts_with_desire",
    col("phrase").rlike("^i like|^i want")
  )
  .show(truncate = false)
```

```
+------------------+------------------+
|phrase            |starts_with_desire|
+------------------+------------------+
|i like tacos      |true              |
|i want love       |true              |
|pie is what i like|false             |
|pizza pizza       |false             |
|you like pie      |false             |
+------------------+------------------+
```

We can also append an `ends_with_food` column using regular expresssions.

```
val foods = List("tacos", "pizza", "pie")
df
  .withColumn(
    "ends_with_food",
    col("phrase").rlike(foods.map(_ + "$").mkString("|"))
  )
  .show(truncate = false)
```

```
+------------------+--------------+
|phrase            |ends_with_food|
+------------------+--------------+
|i like tacos      |true          |
|i want love       |false         |
|pie is what i like|false         |
|pizza pizza       |true          |
|you like pie      |true          |
+------------------+--------------+
```

Matching strings that contain regex characters

Suppose we want to find all the strings that contain the substring `"fun|stuff"`. We don't want all strings that contain fun or stuff—we want all strings that match the substring `fun|stuff` exactly.

The approch we've been using won't work as desired, because it will match all strings that contain `fun` or `stuff`.

```
df
  .withColumn(
    "contains_fun_pipe_stuff",
    col("phrase").rlike("fun|stuff")
  )
  .show(truncate = false)
```

```
+------------------+-----------------------+
|phrase            |contains_fun_pipe_stuff|
+------------------+-----------------------+
|fun|stuff         |true                   |
|dancing is fun    |true                   |
|you have stuff    |true                   |
|where is fun|stuff|true                   |
|something else    |false                  |
+------------------+-----------------------+
```

We can use the `java.util.regex.Pattern` to quote the regular expression and properly match the `fun|stuff` string exactly.

```
import java.util.regex.Pattern

df
  .withColumn(
    "contains_fun_pipe_stuff",
    col("phrase").rlike(Pattern.quote("fun|stuff"))
  )
  .show(truncate = false)
```

```
+------------------+----------------------+
|phrase            |contains_fun_pipe_stuff|
+------------------+----------------------+
|fun|stuff         |true                   |
|dancing is fun    |false                  |
|you have stuff    |false                  |
|where is fun|stuff|true                   |
|something else    |false                  |
+------------------+----------------------+
```

`Pattern.quote("fun|stuff")` returns `"\Qfun|stuff\E"`.

The `Pattern.quote()` method wraps the string in `\Q` and `\E` to turn the text into a regexp literal, as described in this Stackoverflow thread.

Alternatively, we can escape the pipe character in the regexp with `\\`.

```
df
  .withColumn(
    "contains_fun_pipe_stuff",
    col("phrase").rlike("fun\\|stuff")
  )
  .show(truncate = false)
```

```
+------------------+----------------------+
|phrase            |contains_fun_pipe_stuff|
+------------------+----------------------+
|fun|stuff         |true                   |
|dancing is fun    |false                  |
|you have stuff    |false                  |
|where is fun|stuff|true                   |
|something else    |false                  |
+------------------+----------------------+
```

Abstracting multiple pattern match criteria to CSV files

You may want to store multiple string matching criteria in a separate CSV file rather than directly in the code. Let's create a CSV that matches all strings that start with `coffee`, end with `bread` or contain `nice|person`. Here's the content of the `random_matches.csv` file.

```
^coffee
bread$
nice\\|person
```

The pipe character in the CSV file needs to be escaped with `\\`.

Here's the how to use the CSV file to match strings that match at least one of the regexp criteria.

```
val df = List(
  ("coffee is good"),
  ("i need coffee"),
  ("bread is good"),
  ("i need bread"),
  ("you're a nice|person"),
  ("that is nice")
).toDF("phrase")

val weirdMatchesPath = new java.io.File(s"./src/test/resources/ra

val weirdMatchesDF = spark
  .read
  .option("header", "true")
  .option("charset", "UTF8")
  .csv(weirdMatchesPath)

val matchString = DataFrameHelpers.columnToArray[String](
  weirdMatchesDF,
  "match_criteria"
).mkString("|")

df
  .withColumn(
    "weird_matches",
    col("phrase").rlike(matchString)
  )
  .show(truncate = false)
```

```
+--------------------+-------------+
|phrase              |weird_matches|
+--------------------+-------------+
|coffee is good      |true         |
|i need coffee       |false        |
|bread is good       |false        |
|i need bread        |true         |
|you're a nice|person|true         |
|that is nice        |false        |
+--------------------+-------------+
```

Next steps

> "Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.—Jamie Zawinski"

Using regular expressions is controversial to say the least. Regular expressions are powerful tools for advanced string matching, but can create code bases that are difficult to maintain. Thoroughly testing regular expression behavior and documenting the expected results in comments is vital, especially when multiple regexp criteria are chained together.

Spark's `rlike` method allows for powerful string matching. You'll be rewarded with great results if you can learn to use these tools effectively.

Subscribe to get future posts from Matthew Powers

Email*

protected by reCAPTCHA
Privacy - Terms

Submit

Create your own free form with HubSpot

Posted in Apache Spark

1 COMMENT

Sinister Penguin
9 months ago Permalink

Great post – Thank You!

However, its not immediately clear why you are pipeline delim-iting the list here:

col(“phrase”).rlike(animals.mkString(“|”))

Took me a bit more Googling to work out it’s a regEx Logical Or, it would be great to make this a bit clearer for Spark Noobs like me!

Reply

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

Previous Post: Speaking Slack Notifications from Spark
Next Post: Building Spark JAR Files with SBT

Search ...

RECENT POSTS

- Different ways to write CSV files with Dask
- Writing Dask DataFrame to a Single CSV File
- Registering Native Spark Functions
- select and add columns in PySpark
- Filtering PySpark Arrays and DataFrame Array Columns

RECENT COMMENTS

- mrpowers on Exploring DataFrames with summary and describe
- carlo sancassani on Calculating Week Start and Week End Dates with Spark
- Andrew on Exploring DataFrames with summary and describe
- Puterdo Borato on Scala is a Maintenance Nightmare
- Hartmut on Scala is a Maintenance Nightmare

ARCHIVES

- May 2021
- April 2021
- March 2021
- February 2021
- January 2021
- December 2020
- November 2020
- October 2020
- September 2020

- August 2020
- July 2020
- June 2020
- April 2020
- March 2020
- January 2020
- December 2019
- November 2019
- October 2019
- September 2019
- August 2019
- July 2019
- April 2019
- March 2019
- February 2019
- January 2019
- December 2018
- October 2018
- September 2018
- July 2018
- May 2018
- April 2018
- October 2017
- January 2017

CATEGORIES

- Apache Spark
- AWS
- Dask
- Delta Lake
- Emacs
- github
- Golang
- Java
- OSS
- PyArrow
- PySpark
- Python
- Scala

- Spark 3
- sqlite
- Unix

META

- Log in
- Entries feed
- Comments feed
- WordPress.org