Copy of pmcid_text

May 5, 2022

[4]:
```python
import json
import pandas as pd
import random
import os
import zipfile
```

[5]:
```python
import os
os.environ["KAGGLE_CONFIG_DIR"] = "/home/jovyan/"
!chmod 600 "/home/jovyan/pmc_json.zip"
```

[7]:
```python
with zipfile.ZipFile("/home/jovyan/pmc_json.zip","r") as zf:
    zf.extractall('ppmc')
```

[8]:
```python
!pip3 install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/spark-3.2.1-bin-
hadoop3.2/python (3.2.1)
Requirement already satisfied: py4j==0.10.9.3 in /opt/conda/lib/python3.9/site-
packages (from pyspark) (0.10.9.3)
```

[9]:
```python
from pyspark.sql.functions import lit
from pyspark.sql.types import (
    ArrayType,
    IntegerType,
    MapType,
    StringType,
    StructField,
    StructType,
)


def generate_cord19_schema():

    author_fields = [
        StructField("first", StringType()),
        StructField("middle", ArrayType(StringType())),
        StructField("last", StringType()),
        StructField("suffix", StringType()),
  ]

  authors_schema = ArrayType(
    StructType(
        author_fields
        + [
            # Uncomment to cast field into a JSON string. This field is not
            # well-specified in the source.
            StructField(
                "affiliation",
                StructType(
                    [
                        StructField("laboratory", StringType()),
                        StructField("institution", StringType()),
                        StructField(
                            "location",
                            StructType(
                                [
                                    StructField("settlement", StringType()),
                                    StructField("country", StringType()),
                                ]
                            ),
                        ),
                    ]
                ),
            ),
            StructField("email", StringType()),
        ]
    )
)

# used in `section_schema` for citations, references, and equations
  spans_schema = ArrayType(
    StructType(
        [
            # character indices of inline citations
            StructField("start", IntegerType()),
            StructField("end", IntegerType()),
            StructField("text", StringType()),
            StructField("ref_id", StringType()),
        ]
    )
)

# A section of the paper, which includes the abstract, body, and back matter.
  section_schema = ArrayType(
    StructType(
        [
            StructField("text", StringType()),
            StructField("cite_spans", spans_schema),
            StructField("ref_spans", spans_schema),
            # While equations don't appear in the abstract, but appear here
            # for consistency
            StructField("eq_spans", spans_schema),
            StructField("section", StringType()),
        ]
    )
)

  bib_schema = MapType(
    StringType(),
    StructType(
        [
            StructField("ref_id", StringType()),
            StructField("title", StringType()),
            StructField("authors", ArrayType(StructType(author_fields))),
            StructField("year", IntegerType()),
            StructField("venue", StringType()),
            StructField("volume", StringType()),
            StructField("issn", StringType()),
            StructField("pages", StringType()),
            StructField(
                "other_ids",
                StructType([StructField("DOI", ArrayType(StringType()))]),
            ),
        ]
    ),
    True,
)

# Can be one of table or figure captions
  ref_schema = MapType(
    StringType(),
    StructType(
        [
            StructField("text", StringType()),
            # Likely equation spans, not included in source schema, but
            # appears in JSON
            StructField("latex", StringType()),
            StructField("type", StringType()),
        ]
    ),
)
```

```
    return StructType(
      [
          StructField("paper_id", StringType()),
          StructField(
              "metadata",
              StructType(
                  [
                      StructField("title", StringType()),
                      StructField("authors", authors_schema),
                  ]
              ),
              True,
          ),
          StructField("body_text", section_schema),
          StructField("bib_entries", bib_schema),
          StructField("ref_entries", ref_schema),
          StructField("back_matter", section_schema),
      ]
  )
```

[10]:
```
def extract_dataframe_kaggle(spark):
    """Extract a structured DataFrame from the semi-structured document dump.

    It should be fairly straightforward to modify this once there are new
    documents available. The date of availability (`crawl_date`) and `source`
    are available as metadata.
    """
    base = "ppmc/pmc_json"

    dataframe = None

    path = f"{base}/"
    df = (spark.read.json(path, schema=generate_cord19_schema(), multiLine=True)
        )
    if not dataframe:
        dataframe = df
    else:
        dataframe = dataframe.union(df)
    return dataframe
```

[11]:
```
from pyspark.sql import SparkSession

MAX_MEMORY = "20g"
spark = SparkSession \
    .builder \
    .appName("sparkdf") \
    .config("spark.executor.memory", MAX_MEMORY) \
    .config("spark.driver.memory", MAX_MEMORY) \
    .config("spark.memory.offHeap.enabled",True)\
    .config("spark.memory.offHeap.size","80g")\
    .config("spark.ui.port", "4040") \
    .getOrCreate()
spark
```

[11]:
```
<pyspark.sql.session.SparkSession at 0xffff4da32b50>
```

[12]:
```
df = extract_dataframe_kaggle(spark)
```

[13]:
```
from pyspark.conf import SparkConf
conf = SparkConf()
conf.set("spark.driver.memory","15g")
```

[13]:
```
<pyspark.conf.SparkConf at 0xffff4c457d00>
```

[14]:
```
from pyspark.sql import Window
from pyspark.sql import functions as F

title = (
    df.withColumn("title", F.col("metadata").getField("title"))
    .select("paper_id", "title")
)

title.show(5)
```

```
+----------+--------------------+
|  paper_id|               title|
+----------+--------------------+
|PMC8206995|Timing of surgery…|
|PMC7111423|     Poster Sessions|
|PMC7122603|Cardiovascular Ac…|
|PMC7162159|             Posters|
|PMC7130089|             Posters|
+----------+--------------------+
only showing top 5 rows
```

[15]:
```
titlerdd = title.rdd.map(lambda row: row['title'])
```

[16]:
```
titlerdd
```

[16]:
```
PythonRDD[10] at RDD at PythonRDD.scala:53
```

[17]:
```
titlelist=[titlerdd.collect()]
```

[18]:
```
paperidrdd=title.select("paper_id").rdd.flatMap(lambda x: x).collect()
```

[19]:
```
paperidlist=[paperidrdd]
```

[20]:
```
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA

sia = SIA()
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /home/jovyan/nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```

[21]:
```
results=[]
for lines,paperid in zip(titlelist,paperidlist):

    for line,id in zip(lines,paperid) :

        try:
            pol_score = sia.polarity_scores(line)
            pol_score['title'] = line
            pol_score['paper_id']=id
            results.append(pol_score)
        except:
            print('Skipped')
```

[22]:
```
import numpy as np
import pandas as pd
df_result = pd.DataFrame(results)

df_result['label'] = np.zeros(len(df_result)).tolist()
df_result.loc[df_result['compound'] > 0.05, 'label'] = 1
df_result.loc[df_result['compound'] < -0.05 , 'label'] = -1
```

[23]:
```
df_result['label'].value_counts() # title_results
```

[23]:
```
 0.0    150556
 1.0     68399
-1.0     67274
Name: label, dtype: int64
```

[24]:
```
df_result.to_csv('results.csv')
```

[25]:
```
spark.stop()
```

[ ]:
```

```

```
[ ]:
```

```
[ ]:
```