# lda

August 6, 2019

## 1 1. Settings

[1]:
```
import os
import findspark

findspark.init()

from pyspark.sql import *
from pyspark import SparkConf
from pyspark.sql.functions import *
from pyspark.sql import functions as F
```

[2]:
```
local = "local[*]"

appName = "Scientific papers analysis app"

configLocale = SparkConf().setAppName(appName).setMaster(local). \
set("spark.executor.memory", "4G"). \
set("spark.driver.memory", "4G"). \
set("spark.sql.catalogImplementation", "in-memory")

spark = SparkSession.builder.config(conf=configLocale).getOrCreate()
sc = spark.sparkContext
sc.setLogLevel("ERROR")

spark
```

[2]:
```
<pyspark.sql.session.SparkSession at 0x146e1ff93c8>
```

[3]:
```
print("Id : ", sc.applicationId)
print("Version : ", sc.version)
```

```
Id :  local-1565094004468
Version :  2.4.3
```

[4]:
```
# https://medium.com/@connectwithghosh/
 ↪topic-modelling-with-latent-dirichlet-allocation-lda-in-pyspark-2cb3ebd5678e
import pandas as pd
```

```
import pyspark
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
```

[6]:
```
# For building the model
from pyspark.ml.feature import CountVectorizer, HashingTF, IDF
from pyspark.mllib.linalg import Vector, Vectors
#from pyspark.mllib.clustering import LDA, LDAModel
from pyspark.ml.clustering import LDA
```

## 2 2. Data

[8]:
```
path = "./Data/DataMicroTAS/"
filename = "Datas_MicroTAS2018.csv"
# Reading the data
data = sqlContext.read.format("csv")\
   .options(header='true', inferschema='true') \
   .load(os.path.realpath(path + filename))

#print(type(data), dir(data))
data.show(1)
```

```
+---------+--------------------+-----------+----+--------------------+--------
-----------+--------------------+
|  filename|               title|     author|date|            abstract|
keywords|                text|
+---------+--------------------+-----------+----+--------------------+--------
-----------+--------------------+
|PG0001.pdf|ai based personal...|chih-ming ho|2018|['cancer',
'patie...|['personalized', ...|['current', 'drug...|
+---------+--------------------+-----------+----+--------------------+--------
-----------+--------------------+
only showing top 1 row
```

[9]:
```
selection = "text"
data2 = data.select("filename", F.regexp_replace(F.col(selection), "[\[\]']",␣
 ↪"").alias(selection)) # [\$#,]
data2 = data2.select("filename", split(col(selection), ",\s*").alias(selection))
data2.show(1)
data2
```

```
+---------+--------------------+
|  filename|                text|
+---------+--------------------+
|PG0001.pdf|[current, drug, d...|
```

```
+----------+--------------------+
only showing top 1 row
```

[9]: DataFrame[filename: string, text: array<string>]

[111]:
```
#tokens = data.select("filename", "text")
#tokens.show(2)
```

## 3 3. LDA algorithm

[10]:
```
#df_txts = sqlContext.createDataFrame(tokens, ["list_of_words",'index'])
df_txts = data2.select("filename", "text")
df_txts.show(2)
df_txts
```

```
+----------+--------------------+
|  filename|                text|
+----------+--------------------+
|PG0001.pdf|[current, drug, d...|
|PG0004.pdf|[inerti, microflu...|
+----------+--------------------+
only showing top 2 rows
```

[10]: DataFrame[filename: string, text: array<string>]

[47]:
```
# https://github.com/vsmolyakov/pyspark/blob/master/lda.py
# TF
num_features = 8000
cv = CountVectorizer(inputCol="text", outputCol="raw_features",␣
 ↪vocabSize=num_features, minDF=2.0)
cvmodel = cv.fit(df_txts)

vocab = cvmodel.vocabulary

result_cv = cvmodel.transform(df_txts)
#result_cv = result_cv.drop('text')
result_cv.select("raw_features").show(2)
```

```
+--------------------+
|        raw_features|
+--------------------+
|(8000,[0,1,2,4,7,...|
|(8000,[1,2,3,4,5,...|
+--------------------+
```

```
only showing top 2 rows
```

[38]:
```
# Hashting TF
# TF: Both HashingTF and CountVectorizer can be used to generate the term␣
 ↪frequency vectors.
# HashingTF is a Transformer which takes sets of terms and converts those sets␣
 ↪into fixed-length feature vectors.
# https://spark.apache.org/docs/2.2.0/ml-features.html

#hashingTF = HashingTF(inputCol="raw_features", outputCol="tf_features",␣
 ↪numFeatures=num_features)
#result_hashing = hashingTF.transform(result_cv)
#result_hashing = newsgroups.drop('raw_features')
```

[48]:
```
# IDF
idf = IDF(inputCol="raw_features", outputCol="features")
idfModel = idf.fit(result_cv)
result_tfidf = idfModel.transform(result_cv)
#result_tfidf = result_tfidf.drop('raw_features')
result_cv.select("raw_features").show(2)
```

```
+--------------------+
|        raw_features|
+--------------------+
|(8000,[0,1,2,4,7,...|
|(8000,[1,2,3,4,5,...|
+--------------------+
only showing top 2 rows
```

[49]:
```
from pyspark.ml.clustering import LDA
num_topics = 10

lda = LDA(k=num_topics, featuresCol="features", seed=0)
model = lda.fit(result_tfidf)
```

[50]:
```
topics = model.describeTopics()
topics.show()

model.topicsMatrix()
```

```
+-----+--------------------+--------------------+
|topic|         termIndices|         termWeights|
+-----+--------------------+--------------------+
|    0|[279, 324, 1273, ...|[0.05616231648378...|
|    1|[298, 3550, 130, ...|[0.01853075520755...|
|    2|[1752, 1883, 1894...|[0.01487765424661...|
```

```
|    3|[1254, 278, 590, ...|[0.01200005176723...|
|    4|[29, 69, 66, 363,...|[0.01243888924043...|
|    5|[14, 254, 248, 29...|[0.02229931830743...|
|    6|[380, 662, 500, 6...|[0.00915894636570...|
|    7|[0, 14, 52, 445, ...|[0.00791069041898...|
|    8|[74, 177, 14, 242...|[0.00589018865339...|
|    9|[725, 685, 1019, ...|[0.01814091710135...|
+-----+--------------------+--------------------+
```

[50]:
```
DenseMatrix(8000, 10, [1.4247, 2.0198, 1.666, 1.2416, 4.0678, 5.435, 3.9712,
1.1773, ..., 2.9422, 0.59, 0.5267, 0.596, 0.6037, 0.467, 0.5371, 0.5065], 0)
```

[51]:
```
topics_rdd = topics.rdd

topics_words = topics_rdd \
   .map(lambda row: row['termIndices']) \
   .map(lambda idx_list: [vocab[idx] for idx in idx_list]) \
   .collect()

for idx, topic in enumerate(topics_words):
    print("topic: ", idx)
    print("----------")
    for word in topic:
        print(word)
    print("----------")
```

```
topic:  0
----------
fals
true
adob
dental
temp
wind
rpa
sg
pdf
janu
----------
topic:  1
----------
oxygen
p
water
hair
permeat
infant
```

```
micro
sperm
hydrostat
ga
----------
topic:  2
----------
noa
micromesh
protoplast
inkjet
text
plant
pnipaam
silica
adamski
rgo
----------
topic:  3
----------
cilia
nanoparticl
nanochannel
cell
ip
motil
cytoplasm
particl
microtubul
motor
----------
topic:  4
----------
particl
dna
electrod
imped
sensor
sort
frequenc
electr
acoust
focus
----------
topic:  5
----------
droplet
pcr
```

```
bacteria
digit
dmf
bacteri
target
polydispers
rjp
nucleic
----------
topic:  6
----------
crystal
subdoc
ctc
dna
lfa
np
cancer
captur
heat
embryo
----------
topic:  7
----------
cell
droplet
cultur
fiber
hydrogel
membran
fig
tissu
spheroid
lipid
----------
topic:  8
----------
blood
magnet
droplet
antibodi
dilut
detect
assay
sampl
plasma
reagent
----------
```

```
topic:  9
----------
exosom
vesicl
atp
ev
zebrafish
subtyp
isol
larva
freestand
guv
----------
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: