midterm_am12180

November 10, 2024

Midterm Q2~4 - Ahhyun Moon (am12180@nyu.edu)

[2]:
```
import os
import pyspark

conf = pyspark.SparkConf()
conf = conf.setAppName("<my-app-name>")
conf.set('spark.ui.proxyBase', '/user/' + os.environ['JUPYTERHUB_USER'] + '/
 ↪proxy/4040') ## to setup SPARK UI
conf = conf.set('spark.jars', os.environ['GRAPHFRAMES_PATH']) ## graphframes in␣
 ↪spark configuration
sc = pyspark.SparkContext(conf=conf)
spark = pyspark.SQLContext(sc)
spark
```

```
24/11/10 14:19:04 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
/opt/conda/envs/bigdata-spark/lib/python3.11/site-
packages/pyspark/sql/context.py:113: FutureWarning: Deprecated in 3.0.0. Use
SparkSession.builder.getOrCreate() instead.
  warnings.warn(
```

[2]:
```
<pyspark.sql.context.SQLContext at 0x7b90f41fa350>
```

[3]:
```
# Import relevant functinos from Pyspark SQL Library
import pyspark.sql.functions as F
from pyspark.sql.functions import udf
from pyspark.sql import Row
from pyspark.sql.types import IntegerType, DoubleType, FloatType, ArrayType
from pyspark.sql.window import Window
from pyspark.ml.feature import Tokenizer, StopWordsRemover, RegexTokenizer,␣
 ↪NGram
```

1 Question 2

1.1 Used same logic from HW2 to get bigram counts from HW1 Data

[3]:
```
corpus = spark.read.text("../am12180-hw2/data/hw1text/*.txt")\
    .withColumn("file", F.lcase(F.regexp_replace("value", r"[^0-9a-z]", " ")))\
    .withColumn("text", F.regexp_replace("file", r"  +", " "))

# corpus.limit(5).show()
tokenizer = RegexTokenizer(outputCol="words", inputCol="file", pattern=r" +")

bigram = NGram(n=2)
bigram.setInputCol("words")
bigram.setOutputCol("bigrams")

bigram_count = bigram\
                    .transform(tokenizer.transform(corpus))\
                    .select(F.explode("bigrams").alias("bigram"))\
                    .where(F.length("bigram") > 0)\
                    .groupBy("bigram").count()\
                    .orderBy(F.desc("count"))
bigram_count.show(10)
```

```
24/11/10 06:18:57 WARN GarbageCollectionMetrics: To enable non-built-in garbage
collector(s) List(G1 Concurrent GC), users should configure it(them) to
spark.eventLog.gcMetrics.youngGenerationGarbageCollectors or
spark.eventLog.gcMetrics.oldGenerationGarbageCollectors

+-------+-----+
| bigram|count|
+-------+-----+
| of the|17217|
| in the|12045|
|   p he|10876|
| to the| 8227|
|    n t| 5368|
|for the| 5328|
| on the| 4806|
|  to be| 4522|
|will be| 4171|
|and the| 3881|
+-------+-----+
only showing top 10 rows
```

1.2 Get top 10 trigram counts

[4]:
```
trigram = NGram(n=3)
trigram.setInputCol("words")
trigram.setOutputCol("trigrams")

trigram_count = trigram\
                    .transform(tokenizer.transform(corpus))\
                    .select(F.explode("trigrams").alias("trigram"))\
                    .where(F.length("trigram") > 0)\
                    .groupBy("trigram").count()\
                    .orderBy(F.desc("count"))\
                    .limit(10)
trigram_count.show()
```

```
[Stage 5:>                                                          (0 + 2) / 2]
```

```
+------------------+-----+
|           trigram|count|
+------------------+-----+
|           lt p gt| 1929|
|            do n t| 1450|
|     the spread of| 1182|
|           p gt lt| 1037|
|           gt lt p| 1023|
|of the coronavirus|  862|
|        as well as|  830|
|     the number of|  821|
|        one of the|  811|
|     spread of the|  779|
+------------------+-----+
```

1.3 Compute conditional probability of the third word of each trigram

[5]:
```
# Extract bigram from each trigram in trigram_count
trigram_count = trigram_count.withColumn("bigram",\
                                         F.concat_ws(" ", F.split(F.
 ↪col("trigram"), " ").getItem(0),\
                                                     F.split(F.
 ↪col("trigram"), " ").getItem(1)))

# Join trigram df with bigram df on the extracted bigram
trigram_with_bigram_count = trigram_count.join(
    bigram_count.select(F.col("bigram").alias("bigram_word"), F.col("count").
 ↪alias("bigram_count")),
```

```
    trigram_count["bigram"] == F.col("bigram_word"),
    "left" )

# Calculate the conditional probability for each trigram
trigram_cond_prob = trigram_with_bigram_count.withColumn("probability", F.
 ↪round(F.col("count") / F.col("bigram_count"), 5))

print("Result for Q2:")

# Show the result
trigram_cond_prob.select("trigram", "count", "bigram", "bigram_count",␣
 ↪"probability")\
                 .orderBy(F.col("count").desc())\
                 .show()
```

Result for Q2:

[Stage 15:> (0 + 2) / 2]

```
+------------------+-----+----------+------------+-----------+
|           trigram|count|    bigram|bigram_count|probability|
+------------------+-----+----------+------------+-----------+
|           lt p gt| 1929|      lt p|        1931|    0.99896|
|            do n t| 1450|      do n|        1451|    0.99931|
|     the spread of| 1182|the spread|        1288|     0.9177|
|           p gt lt| 1037|      p gt|        1941|    0.53426|
|           gt lt p| 1023|     gt lt|        1804|    0.56707|
|of the coronavirus|  862|    of the|       17217|    0.05007|
|        as well as|  830|   as well|        1134|    0.73192|
|     the number of|  821|the number|         869|    0.94476|
|        one of the|  811|    one of|        1322|    0.61346|
|     spread of the|  779| spread of|        1417|    0.54975|
+------------------+-----+----------+------------+-----------+
```

2 Question 3

[6]:
```
bakeryData = spark\
  .read\
  .option("inferSchema", "true")\
  .option("header", "true")\
  .csv("../am12180-hw2/data/Bakery.csv")
bakeryData2 = bakeryData.withColumn("hour-period", F.date_format("Time", "HH").
 ↪cast(IntegerType()))

# Add DayPart by categorizing Hour-period in to morning, afternoon, and evening
```

```
bakeryData2 = bakeryData2.withColumn("dayPart",
                                     F.when(F.col("hour-period").between(6,␣
 ↪10), "morning")\
                                     .when(F.col("hour-period").between(11,␣
 ↪13), "noon")\
                                     .when(F.col("hour-period").between(14,␣
 ↪16), "afternoon")\
                                     .when(F.col("hour-period").between(17,␣
 ↪23), "evening")\
                                     .otherwise("evening")) # 0 - 5 am

# Group by same DayPart, Item and aggregate transaction as total sum
grouped_bakeryData2 = bakeryData2.groupBy("dayPart", "Item").agg(F.count("*").
 ↪alias("count"))

# Create a window specification to order items within each and Daypart
window_spec = Window.partitionBy("dayPart").orderBy(F.desc("count"))

# Give row number for ranking
ranked_bakeryData = grouped_bakeryData2.withColumn("rank", F.row_number().
 ↪over(window_spec))

# Filter only top 3 items in the ranking
top3_items = ranked_bakeryData.filter(F.col("rank") <= 3)

# Put top 3 items into a single column
grouped_top3_items = top3_items.groupBy("dayPart").agg(F.collect_list(F.
 ↪col("Item")).alias("top3"))

print("Result for Q3:")
# Show result
grouped_top3_items.show(truncate=False)
```

```
Result for Q3:
[Stage 18:>                                                         (0 + 1) / 1]
+---------+-----------------------+
|dayPart  |top3                   |
+---------+-----------------------+
|afternoon|[Coffee, Bread, Tea]   |
|evening  |[Coffee, Bread, Tea]   |
|morning  |[Coffee, Bread, Pastry]|
|noon     |[Coffee, Bread, Tea]   |
+---------+-----------------------+
```

3 Question 4

[7]:
```
# %conda install datasketch
```

[7]:
```
from pyspark.ml.feature import Tokenizer, HashingTF, IDF, MinHashLSH
from pyspark.ml.linalg import Vectors, VectorUDT, SparseVector
from pyspark.sql.functions import monotonically_increasing_id
```

[5]:
```
huffpost_df = spark.read.json("./Huffpost.json")
#  .sample(fraction=0.5, seed=3)
huffpost_df.show(5)
huffpost_df.count()
```

```
+--------------------+---------+----------+--------------------+----------------
----+--------------------+
|             authors| category|      date|            headline|
link|   short_description|
+--------------------+---------+----------+--------------------+----------------
----+--------------------+
|Carla K. Johnson, AP|U.S. NEWS|2022-09-23|Over 4 Million
Am…|https://www.huffp…|Health experts sa…|
|      Mary Papenfuss|U.S. NEWS|2022-09-23|American
Airlines…|https://www.huffp…|He was subdued by…|
|       Elyse Wanshel|   COMEDY|2022-09-23|23 Of The
Funnies…|https://www.huffp…|"Until you have a…|
|    Caroline Bologna|PARENTING|2022-09-23|The Funniest
Twee…|https://www.huffp…|"Accidentally put…|
|      Nina Golgowski|U.S. NEWS|2022-09-22|Woman Who Called
…|https://www.huffp…|Amy Cooper accuse…|
+--------------------+---------+----------+--------------------+----------------
----+--------------------+
only showing top 5 rows
```

[5]:
```
209527
```

3.1 Find top 5 similar articles using Spark ML MinHshLSH Jaccard Distance

[8]:
```
base = "Kitten Born With Twisted Arms And Legs Finds A Mom Who Knows She\u2019s␣
 ↪Perfect"
query = spark.createDataFrame([("", "", "", "", "", base)])
```

```
df = huffpost_df.union(query)
cleaned_df = df.select("short_description").distinct().withColumn("id",␣
 ↪monotonically_increasing_id())
print(cleaned_df.count())
cleaned_df.show(5)
```

```
187023
```

```
+--------------------+---+
|   short_description| id|
+--------------------+---+
|White House offic…|  0|
|The director told…|  1|
|Residents of Miss…|  2|
|The GOP is contin…|  3|
|The lawsuit looks…|  4|
+--------------------+---+
only showing top 5 rows
```

3.1.1 Tokenized the short description

[9]:
```
# Tokenize the text
tokenizer = RegexTokenizer(pattern="\\W", inputCol="short_description",␣
 ↪outputCol="words")
wordsData = tokenizer.transform(cleaned_df).filter( F.size(F.col("words")) != 0␣
 ↪)
wordsData.show(5)
```

```
[Stage 16:>                                                         (0 + 1) / 1]
```

```
+--------------------+---+--------------------+
|   short_description| id|               words|
+--------------------+---+--------------------+
|White House offic…|  0|[white, house, of…|
|The director told…|  1|[the, director, t…|
|Residents of Miss…|  2|[residents, of, m…|
|The GOP is contin…|  3|[the, gop, is, co…|
|The lawsuit looks…|  4|[the, lawsuit, lo…|
+--------------------+---+--------------------+
only showing top 5 rows
```

3.1.2 Mapped the short description tokens to sparse vectors

[10]:
```
# HashingTF to create feature vectors
hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures",␣
 ↪numFeatures=2048)
featurizedData = hashingTF.transform(wordsData)
featurizedData.head().rawFeatures
```

[10]:
```
SparseVector(2048, {131: 1.0, 251: 1.0, 271: 1.0, 344: 1.0, 583: 1.0, 611: 1.0,
779: 1.0, 812: 1.0, 952: 1.0, 999: 1.0, 1051: 1.0, 1238: 1.0, 1292: 1.0, 1444:
1.0, 1449: 1.0, 1626: 2.0, 1649: 1.0, 1681: 3.0, 1805: 1.0, 1843: 1.0, 1847:
1.0, 1876: 1.0, 1889: 1.0, 1999: 2.0, 2045: 1.0})
```

3.1.3 Compute the inverse document frequency weights based on the vectors

[11]:
```
# Fit the IDF model and transform the original feature vectors
idf = IDF(minDocFreq=3,inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData)
rescaledData.head().features
```

[11]:
```
SparseVector(2048, {131: 3.1348, 251: 4.6341, 271: 3.4098, 344: 4.4232, 583:
3.612, 611: 0.8947, 779: 0.9645, 812: 4.0275, 952: 0.8382, 999: 0.7505, 1051:
3.4936, 1238: 3.7234, 1292: 2.8062, 1444: 2.6402, 1449: 4.444, 1626: 6.9949,
1649: 5.8594, 1681: 1.3485, 1805: 6.4889, 1843: 4.2229, 1847: 2.1811, 1876:
5.1804, 1889: 4.3451, 1999: 1.9004, 2045: 2.1874})
```

3.1.4 Use MinHashLSH to approximate the similarity distance between short descriptions (features)

[12]:
```
# Create and fit the MinHashLSH model to the feature vectors
mh = MinHashLSH(inputCol="features", outputCol="hashes", seed=12345)
model = mh.fit(rescaledData)
# Transform the feature data to include hash values
transformedData = model.transform(rescaledData)
transformedData.head()
```

[12]:
```
Row(short_description="White House officials say the crux of the president's
visit to the U.N. this year will be a full-throated condemnation of Russia and
its brutal war.", id=0, words=['white', 'house', 'officials', 'say', 'the',
'crux', 'of', 'the', 'president', 's', 'visit', 'to', 'the', 'u', 'n', 'this',
```

```
'year', 'will', 'be', 'a', 'full', 'throated', 'condemnation', 'of', 'russia',
'and', 'its', 'brutal', 'war'], rawFeatures=SparseVector(2048, {131: 1.0, 251:
1.0, 271: 1.0, 344: 1.0, 583: 1.0, 611: 1.0, 779: 1.0, 812: 1.0, 952: 1.0, 999:
1.0, 1051: 1.0, 1238: 1.0, 1292: 1.0, 1444: 1.0, 1449: 1.0, 1626: 2.0, 1649:
1.0, 1681: 3.0, 1805: 1.0, 1843: 1.0, 1847: 1.0, 1876: 1.0, 1889: 1.0, 1999:
2.0, 2045: 1.0}), features=SparseVector(2048, {131: 3.1348, 251: 4.6341, 271:
3.4098, 344: 4.4232, 583: 3.612, 611: 0.8947, 779: 0.9645, 812: 4.0275, 952:
0.8382, 999: 0.7505, 1051: 3.4936, 1238: 3.7234, 1292: 2.8062, 1444: 2.6402,
1449: 4.444, 1626: 6.9949, 1649: 5.8594, 1681: 1.3485, 1805: 6.4889, 1843:
4.2229, 1847: 2.1811, 1876: 5.1804, 1889: 4.3451, 1999: 1.9004, 2045: 2.1874}),
hashes=[DenseVector([69662265.0])])
```

[13]:
```
baseArticle = transformedData.filter(F.col("short_description") == base)
baseArticle.show()
```

```
[Stage 33:>                                                         (0 + 1) / 1]

+--------------------+----------+--------------------+--------------------+-----
--------------+---------------+
|   short_description|        id|               words|         rawFeatures|
features|         hashes|
+--------------------+----------+--------------------+--------------------+-----
--------------+---------------+
|Kitten Born With …|8590028459|[kitten, born,
wi…|(2048,[34,381,386…|(2048,[34,381,386…|[[1.5298664E8]]|
+--------------------+----------+--------------------+--------------------+-----
--------------+---------------+
```

[22]:
```
similarNews = model.approxSimilarityJoin(transformedData, F.
  ↪broadcast(baseArticle), 1, distCol="JaccardDistance").select(
        F.col("datasetA.id").alias("A_id"),
        F.col("datasetA.short_description").alias("A_description"),
        F.col("datasetB.id").alias("B_id"),
        F.col("datasetB.short_description").alias("B_description"),
        F.col("JaccardDistance")
    ) \
    .filter("A_id != B_id")\
    .withColumn("jaccard", F.col("JaccardDistance").cast(FloatType()))

similarNews.show(5)
```

```
[Stage 88:============================>                             (1 + 1) / 2]

+-----+--------------------+----------+--------------------+------------------+-
---------+
| A_id|       A_description|      B_id|       B_description|   JaccardDistance|
```

```
jaccard|
+-----+--------------------+----------+--------------------+------------------+-
---------+
|76604|What kinds of tho…|8590028459|Kitten Born With …|            0.9375|
0.9375|
|13888|This post first a…|8590028459|Kitten Born With …|            0.9375|
0.9375|
|42409|The former secret…|8590028459|Kitten Born With …|0.8666666666666667|
0.8666667|
|59504|Thomas Lane, J. K…|8590028459|Kitten Born With …|0.9444444444444444|
0.9444444|
|59732|He won't use Dona…|8590028459|Kitten Born With
…|0.8846153846153846|0.88461536|
+-----+--------------------+----------+--------------------+------------------+-
---------+
only showing top 5 rows
```

3.1.5 Find URL link, headline, category, and short description of the 5 most similar items to the Item above (based on the "short_description" field)

[26]:
```
top5 = similarNews.orderBy("jaccard") \
        .select("A_id", "A_description", "jaccard") \
        .limit(5)
```

[30]:
```
result = top5.join(huffpost_df, top5.A_description == huffpost_df.
 ↪short_description, "left")
result_list = result.select("headline", "link", "category",␣
 ↪"short_description", "jaccard").orderBy("jaccard").collect()

for item in result_list:
    print("Link:", item.link)
    print("Headline:", item.headline)
    print("Category:", item.category)
    print("Short Description:", item.short_description)
    print("Jaccard Distance:", item.jaccard)
    print("\n")
```

```
[Stage 157:>                                                        (0 + 2) / 2]
```

```
Link: https://www.huffingtonpost.com/entry/short-haircut-makeover-
video_us_5b9c5c46e4b03a1dcc7e1459
Headline: How A Simple Short Haircut Can Make For A Dramatic Makeover (VIDEO)
Category: STYLE & BEAUTY
Short Description: She's sexy and she knows it!
Jaccard Distance: 0.7647058963775635
```

```
Link: https://www.huffingtonpost.com/entry/cheryl-boone-isaacs-academy-
president-statement-diversity_us_569e3587e4b00f3e9862c429
Headline: Academy President Releases Official Statement On The Oscars' Lack Of
Diversity
Category: ENTERTAINMENT
Short Description: Amid the backlash, Cheryl Boone Isaacs says that she's
"heartbroken and frustrated."
Jaccard Distance: 0.782608687877655


Link: https://www.huffingtonpost.com/entry/camille-cosby-defamation-
deposition_us_56869f2de4b06fa688826fb1
Headline: Camille Cosby Ordered To Testify In Defamation Suit Against Bill Cosby
Category: ENTERTAINMENT
Short Description: She's scheduled to give a deposition Jan. 6.
Jaccard Distance: 0.800000011920929


Link: https://www.huffingtonpost.comhttp://pubx.co/wfVGIs
Headline: Dad Turns Himself Into Real-Life ‘Elf on the Shelf’
Category: WEIRD NEWS
Short Description: A father in Moncton, New Brunswick, knows if his kids are
naughty and nice because he's the Elf on the Shelf who is keeping
Jaccard Distance: 0.8064516186714172


Link: https://www.huffingtonpost.com/entry/drake-goes-above-literally-and-
beyond-with-his-latest-rihanna-pda_us_57c5bcf7e4b0cdfc5ac98321
Headline: Drake Goes Above (Literally) And Beyond With His Latest Rihanna PDA
Category: ENTERTAINMENT
Short Description: He's in love, he's in love, and he doesn't care who knows it!
Jaccard Distance: 0.8095238208770752
```

3.2 Comparison with Datasketch MinHash - Jaccard Similarity

[32]:
```
from datasketch import MinHash, MinHashLSH

# Define base and compute base_minhash using datasketch's MinHash
base = "Kitten Born With Twisted Arms And Legs Finds A Mom Who Knows She’s␣
 ↪Perfect"
base_tokens = base.lower().split(' ')
base_minhash = MinHash()
```

```
# Update the base_minhash with each token
for token in base_tokens:
    base_minhash.update(token.encode('utf8'))

# Set up a tokenizer to split each row of texts into words
regex_tokenizer = RegexTokenizer(inputCol="short_description",␣
 ↪outputCol="splitted_description",
                                  pattern="\\W",  # Split by non-word characters␣
 ↪(punctuation, spaces, etc.)
                                  toLowercase=True ) # Convert text to lowercase
words = regex_tokenizer.transform(huffpost_df)

huffpost_df_splitted = words.select("link", "headline", "category",␣
 ↪"short_description","splitted_description")
# huffpost_df_splitted.show()
# Define UDF with the return type as FloatType
@F.udf(FloatType())
def minHashEncode(splitted_description):
    # Create a MinHash for the current description
    min_hash = MinHash()
    min_hash.update_batch([s.encode('utf-8') for s in splitted_description])
    return base_minhash.jaccard(min_hash)

# Add a column with the Jaccard similarity score
df_with_jaccard = huffpost_df_splitted.withColumn("jaccard", minHashEncode(F.
 ↪col("splitted_description")))
df_jaccard_filtered = df_with_jaccard.filter(F.col("jaccard") > 0)
# df_jaccard_filtered.show()
top_5 = df_jaccard_filtered.orderBy(F.desc("jaccard")).take(5)
for item in top_5:
    print("Link:", item.link)
    print("Headline:", item.headline)
    print("Category:", item.category)
    print("Short Description:", item.short_description)
    print("Jaccard Similarity:", item.jaccard)
    print('\n')
```

```
[Stage 165:============================>                            (1 + 1) / 2]
```

```
Link: https://www.huffingtonpost.com/entry/how-to-make-the-perfect-
c_us_5b9dc24ee4b03a1dcc8c8503
Headline: How to Make the Perfect Chocolate Chip Cookie
Category: FOOD & DRINK
Short Description: A cookie with the perfect combination of fat, flavor, and
comfort. Who needs detox?
Jaccard Similarity: 0.21875
```

```
Link: https://www.huffingtonpost.com/entry/chiles-rellenos-
recipe_us_5b9d7fb8e4b03a1dcc88c9d3
Headline: Recipe Of The Day: Chiles Rellenos
Category: FOOD & DRINK
Short Description: Topped with a tomato-and-avocado salsa.
Jaccard Similarity: 0.203125


Link: https://www.huffingtonpost.com/entry/-westworld-
trailer_us_5767f302e4b015db1bc9d30f
Headline: The 'Westworld' Trailer Looks Like 'Jurassic Park' With Robots
Category: ENTERTAINMENT
Short Description: Life finds a way.
Jaccard Similarity: 0.1796875


Link: https://www.huffingtonpost.com/entry/michael-caine-slams-young-actors-who-
just-want-to-be-rich-and-famous_us_5732292be4b0bc9cb0484b71
Headline: Michael Caine Slams Young Actors Who Just Want To Be 'Rich And Famous'
Category: ENTERTAINMENT
Short Description: "They do a little part on television and everyone knows who
they are. They can't really act."
Jaccard Similarity: 0.1796875


Link: https://www.huffingtonpost.com/entry/maryland-governor-candidate-
breastfeeds-in-new-campaign-ad_us_5ab3a1dce4b0decad0475af7
Headline: Maryland Governor Candidate Breastfeeds In New Campaign Ad
Category: WOMEN
Short Description: "I’m a mom. I’m a woman. And I want to be your next
governor."
Jaccard Similarity: 0.171875
```

[ ]: