Mattia Danese
CS 119 - Big Data
Professor Singh
Quiz 6: Text and Image Processing

Question 1

1.

Q1. Interpreting association rules [15]

The above table, has columns antecedent, consequent, confidence, lift and support.

1. Explain the first row, `[3] [2] 0.8 1.0666666666666667 0.5` in plain English.

   The first row signifies that when people buy the antecedent, item 3 (coke), they will also buy the consequent, item 2 (beer). The confidence of this association is 0.8, which means 80% of people who bought coke also bought beer. The lift of this association, which is the quotient of the probability of someone buying beer given that they also bought coke divided by the probability of someone buying beer, is greater than one; thus, it can be concluded that buying coke increases the probability of buying beer. Finally, the support is 0.5 which indicates that 50% of people bought coke and beer.

2. The first and the third rows have the antecedent and consequent switched, but different confidence values. (Same with second and fourth rows). How do you explain those results?

   The discrepancies in confidence values between rows is due to association not being bi-directional. In other words, row 1 shows that 80% of people who bought coke also bought beer, but this has no indication of the reverse case and certinaly does not mean that 80% of people who bought beer also bought coke. In fact, row 3 shows that only 66.67% of people who bough beer also bought coke. Simply put, more people also bought beer when they bought coke and less people also bought coke when they bought beer. Everything mentioned above applies to rows 2 and 4 which regard milk and beer.

3. What does support = 0.5 for all the rows mean?

   A support equal to 0.5 signifies that 50% of all people bought the ancedent and consequent for that row.

2.

Q2. Interpreting the new association rules [10]

The third row of the result shows a low support (0.375) and a high lift (1.2). What does this line tell us?

The high lift indicates that buying juice greatly increases the probability of also buying coke. The low support indicates that only 37.5% of people ended up buying both juice and coke.

3.

Q3. Association Rules for an Online Retail Dataset [5]

The main part of this exercise involves processing a sampled dataset from a UK-based online retailer. We'll be working with a 8050 record subset.

- Read in the data from the dataset `online_retail_III.csv`. For your convenience, I have already thrown away bad records using `dropna()`.
- There are a couple of wrinkles to keep in mind in case you are curious, though you may not really need them.
  - An invoice represents a shopping cart and it can contain multiple items.
  - Some invoice numbers start with a "C." Invoice number C123456 is to be interpreted as a return of items in invoice 123456. The `inum` column represents the Invoice number as well as the credit (return). In other words, Invoice numbers `123456` and `C123456` would have `inum` == 123456.

```
[1] import pandas as pd

    df_orig = pd.read_csv('https://storage.googleapis.com/singhj-cs-119/online_retail_II.csv')
    df_orig.dropna(inplace=True)
    df_orig.query("StockCode != 'POST'", inplace=True)
    df_orig.query("StockCode != 'M'", inplace=True)
    df_orig
```

	Invoice	StockCode	Description	Quantity	InvoiceDate	Price	Customer ID	Country
0	489434	85048	15CM CHRISTMAS GLASS BALL 20 LIGHTS	12	2009-12-01 07:45:00	6.95	13085.0	United Kingdom
1	489434	79323P	PINK CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
2	489434	79323W	WHITE CHERRY LIGHTS	12	2009-12-01 07:45:00	6.75	13085.0	United Kingdom
3	489434	22041	RECORD FRAME 7" SINGLE SIZE	48	2009-12-01 07:45:00	2.10	13085.0	United Kingdom
4	489434	21232	STRAWBERRY CERAMIC TRINKET BOX	24	2009-12-01 07:45:00	1.25	13085.0	United Kingdom
...	...	...	...	...	...	...	...	...
1067365	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France
1067366	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
1067367	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
1067368	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
1067369	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France

821230 rows × 8 columns

4.

Q4. Connecting Online Retail Data to FP-Growth [30]

Adapt the DataFrame to look like `df_basket` above.

- `df_orig` is a Pandas DataFrame whereas `df_basket`-equivalent will have to be Spark DataFrames.
- `Invoice` and `StockCode` are strings but FP-Growth needs inputs to be integers. You'd need to map strings to integers before feeding them to FP-Growth and convert the resulting antecedents and consequents back.

Establish the mapping between Invoice IDs, StockCodes and unique integers.

```
import numpy as np
import string


map = {} # contains invoice -> [stockcodes] mappings
map_descriptions = {} # maps int stockcode -> description
map_original_stockcodes = {} # maps (stockcode, description) -> original stockcode string

for data in np.array([df_orig["Invoice"].values, df_orig["StockCode"].values, df_orig["Description"].values]).transpose():
    invoice = data[0].strip(string.ascii_letters).strip()
    stockcode = data[1].strip(string.ascii_letters).strip()

    if not invoice or not stockcode:
      continue

    invoice = int(invoice)
    stockcode = int(stockcode)

    if invoice in map.keys():
      map[invoice].append(stockcode)
    else:
      map[invoice] = [stockcode]

    map_descriptions[stockcode] = data[2]
    map_original_stockcodes[(stockcode, data[2])] = data[1]
```

```
[ ] items = []
    for key, values in map.items():
      items.append((key, list(set(values))))

    df_orig_spark = spark.createDataFrame(items, ["id", "items"])
    df_orig_spark.take(10)
```

```
[Row(id=489434, items=[21871, 22064, 21232, 21523, 85048, 22041, 79323]),
 Row(id=489435, items=[22353, 22195, 22349, 22350]),
 Row(id=489436, items=[22119, 21181, 84596, 48173, 84879, 22194, 21756, 21333, 82582, 22295, 22296, 22107, 21754, 21755, 35004, 22109, 22142, 22111]),
 Row(id=489437, items=[22272, 22145, 22274, 10002, 21912, 84507, 22143, 35400, 20695, 20703, 22111, 22112, 21987, 21989, 21351, 21352, 84970, 20971, 21360, 22130, 21364, 37370, 22271]),
 Row(id=489438, items=[84032, 85183, 21410, 21411, 21252, 20711, 84519, 21033, 21100, 85132, 21329, 84031]),
 Row(id=489439, items=[85216, 16161, 21731, 16169, 20749, 22352, 22065, 22130, 21491, 22064, 21493, 85014, 85232, 84691, 22138, 22139, 22333]),
 Row(id=489440, items=[22349, 22350]),
 Row(id=489441, items=[22321, 22138, 84029, 22111]),
 Row(id=489442, items=[22272, 21888, 85123, 22024, 22025, 22026, 22029, 22031, 22296, 84251, 21916, 21790, 84899, 21428, 21430, 85178, 21955, 22091, 21582, 21586, 21590, 22111, 22271]),
 Row(id=489443, items=[21035, 82001, 20754, 21041, 82580, 22041, 85150])]
```

5.

Q5. Fine-tuning FP-Growth runs [20]

- Set `minConfidence` = 0.75.
- Set `minSupport` such that the total number of association rules is between 10 and 20. (If `minSupport` is small, the number of association rules will increase. As it increases, the number of association rules will decrease.).

```
fpGrowth_spark = FPGrowth(itemsCol="items", minSupport=0.01, minConfidence=0.75)
model_spark = fpGrowth_spark.fit(df_orig_spark)

association_df = model_spark.associationRules.show()
```

```
/content/spark-3.2.1-bin-hadoop3.2/python/pyspark/sql/context.py:125: FutureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
  warnings.warn(
+--------------+----------+------------------+------------------+--------------------+
|    antecedent|consequent|        confidence|              lift|             support|
+--------------+----------+------------------+------------------+--------------------+
|       [22748]|   [22745]|0.7641357027463651| 60.02087734559463|0.010772524369135466|
|[21928, 21929]|   [85099]|0.7969283276450512| 7.857967439981789|0.010635875011387446|
|[22698, 22699]|   [22697]|            0.8832| 38.62504541832669| 0.01251740912817 71|
|       [22745]|   [22748]|0.8461538461538461|60.020877345594634|0.010772524369135466|
|       [82581]|   [82580]|0.7544097693351425|  35.4274055101256|0.012662840484649723|
|[22698, 22697]|   [22699]|0.8440366972477065| 32.76742997590831| 0.01251740912817 71|
|[22411, 22386]|   [85099]|0.8441330998248686| 8.323421546622576|0.010977498405757492|
|       [22697]|   [22699]|0.7709163346613546|29.928730700540015|  0.0176277671494944|
|[22411, 21931]|   [85099]|0.8309352517985612| 8.193286556472316| 0.01052200054659743|
|       [21086]|   [21094]|            0.7872|49.590211764705884|0.011205247335337523|
|       [20679]|   [15056]|0.7923930269413629|26.022732256500646| 0.01138744647900155|
|       [21124]|   [21122]|0.8023255813953488| 47.41387836865004|0.011000273298715496|
|[22697, 22423]|   [22699]|0.8330241187384044| 32.33989655664533| 0.01022592693814339|
|[21931, 22386]|   [85099]|0.8704918032786885|  8.58332676810255|0.012093468160699644|
|       [22698]|   [22697]|0.8154613466334164| 35.66262630276897|0.014894779994534025|
|       [22698]|   [22699]|0.7793017456359103|30.254271483095977|0.014234308098751937|
+--------------+----------+------------------+------------------+--------------------+
```

6.

Q6. Final Association Rules [20]

Present the resulting Association Rules in terms of the original StockCodes and Descriptions, in descending order of lift.

```
from traitlets.utils import descriptions
from pyspark.sql.functions import desc

association_df = model_spark.associationRules.orderBy(desc("lift")).toPandas()

# gets the descriptions for all antecedents and consequents
description_list_ant = []
description_list_cons = []
for i in range(len(association_df["antecedent"])):
  ant_list = []
  cons_list = []

  for ant in association_df["antecedent"][i]:
    ant_list.append(map_descriptions[ant])

  for cons in association_df["consequent"][i]:
    cons_list.append(map_descriptions[cons])

  description_list_ant.append(", ".join(ant_list))
  description_list_cons.append(", ".join(cons_list))

# makes columns id df for descriptions
association_df["antecedent_description(s)"] = description_list_ant
association_df["consequent_description(s)"] = description_list_cons

# gets orignal stockcode strings
original_stockcodes_ant = []
original_stockcodes_cons = []
for i in range(len(association_df["antecedent"])):
  orig_codes_ant = []
  orig_codes_cons = []
  for j in range(len(association_df["antecedent"][i])):
    orig_codes_ant.append(map_original_stockcodes[(association_df["antecedent"][i][j], association_df["antecedent_description(s)"][i].split(", ")[j])])

  for j in range(len(association_df["consequent"][i])):
    orig_codes_cons.append(map_original_stockcodes[(association_df["consequent"][i][j], association_df["consequent_description(s)"][i].split(", ")[j])])

  original_stockcodes_ant.append(orig_codes_ant)
  original_stockcodes_cons.append(orig_codes_cons)

# overwrites previous columns with orignal stockcode strings
association_df["antecedent"] = original_stockcodes_ant
association_df["consequent"] = original_stockcodes_cons

association_df
```

```
/content/spark-3.2.1-bin-hadoop3.2/python/pyspark/sql/context.py:125: FutureWarning: Deprecated in 3.0.0. Use SparkSession.builder.getOrCreate() instead.
  warnings.warn(
```

	antecedent	consequent	confidence	lift	support	antecedent_description(s)	consequent_description(s)
0	[22745]	[22748]	0.846154	60.020877	0.010773	POPPY'S PLAYHOUSE BEDROOM	POPPY'S PLAYHOUSE KITCHEN
1	[22748]	[22745]	0.764136	60.020877	0.010773	POPPY'S PLAYHOUSE KITCHEN	POPPY'S PLAYHOUSE BEDROOM
2	[21086]	[21094]	0.787200	49.590212	0.011205	SET/6 RED SPOTTY PAPER CUPS	SET/6 RED SPOTTY PAPER PLATES
3	[21124]	[21122]	0.802326	47.413878	0.011000	SET/10 BLUE POLKADOT PARTY CANDLES	SET/10 PINK POLKADOT PARTY CANDLES
4	[22698, 22699]	[22697]	0.883200	38.625045	0.012572	PINK REGENCY TEACUP AND SAUCER, ROSES REGENCY ...	GREEN REGENCY TEACUP AND SAUCER
5	[22698]	[22697]	0.815461	35.662626	0.014895	PINK REGENCY TEACUP AND SAUCER	GREEN REGENCY TEACUP AND SAUCER
6	[82581]	[82580]	0.754410	35.427406	0.012663	TOILET METAL SIGN	BATHROOM METAL SIGN
7	[22698, 22697]	[22699]	0.844037	32.767430	0.012572	PINK REGENCY TEACUP AND SAUCER, GREEN REGENCY ...	ROSES REGENCY TEACUP AND SAUCER
8	[22697, 22423]	[22699]	0.833024	32.339897	0.010226	GREEN REGENCY TEACUP AND SAUCER, REGENCY CAKES...	ROSES REGENCY TEACUP AND SAUCER
9	[22698]	[22699]	0.779302	30.254271	0.014234	PINK REGENCY TEACUP AND SAUCER	ROSES REGENCY TEACUP AND SAUCER
10	[22697]	[22699]	0.770916	29.928731	0.017628	GREEN REGENCY TEACUP AND SAUCER	ROSES REGENCY TEACUP AND SAUCER
11	[20679]	[15056BL]	0.792393	26.022732	0.011387	EDWARDIAN PARASOL RED	EDWARDIAN PARASOL BLACK
12	[21931, 22386]	[85099C]	0.870492	8.583327	0.012093	JUMBO STORAGE BAG SUKI, JUMBO BAG PINK POLKADOT	JUMBO BAG BAROQUE BLACK WHITE
13	[22411, 22386]	[85099C]	0.844133	8.323422	0.010977	JUMBO SHOPPER VINTAGE RED PAISLEY, JUMBO BAG P...	JUMBO BAG BAROQUE BLACK WHITE
14	[22411, 21931]	[85099C]	0.830935	8.193287	0.010522	JUMBO SHOPPER VINTAGE RED PAISLEY, JUMBO STORA...	JUMBO BAG BAROQUE BLACK WHITE
15	[21928, 21929]	[85099C]	0.796928	7.857967	0.010636	JUMBO BAG SCANDINAVIAN BLUE PAISLEY, JUMBO BAG...	JUMBO BAG BAROQUE BLACK WHITE

The results are in descending order of lift because a lift greater than one signifies that buying the antecedent increases the probability of buying the consequent. Thus, ordering results in descending order of lift allows for the stronger association rules to print first.

Note: The Colab notebook can be found here.

Question 2

1. 10 million images * 175 Kbytes = 1,750,000,000 Kbytes = 1750 GB
   1750 GB of disk space would be required to store all the images.

2. 10 million images * 1,000 factors * 2 bytes = 20,000,000,000 bytes = 20 GB
   20 GB of disk space would be required to store all the factors of all the images.

3. Factors may be converted to a set of 0/1 matrix elements by having some threshold, $0 < h < 1$, where any factor less than *h* will be set to 0 and any factor greater than *h* will be set to 1. As such, any factor above the threshold of *h* will be deemed "similar" (or more specifically "similar enough") to all other factors above the threshold, reason as to why they will be set to 1. Likewise, any factor below the threshold of *h* will be deemed "not similar" (or again "not similar enough") to all other factors above the threshold, reason as to why they will be set to 0.

4. 10 million images * 1,000 factors * 1 byte = 10,000,000,000 bytes = 10 GB
   The matrix of factors will be about 10 GB in size.

5. Assuming we have *m* number of hash functions, then the minhash matrix will have *m* rows and 1000 columns (one column per factor).

   The total storage required for the minhash matrix will be:
   1000 factors * *m* hash functions * 4 bytes per hash in slot = 4000*m* bytes = 4*m* KB

6. Assuming we have *k* buckets, each band of each column of the minhash matrix will be hashed into one of these *k* buckets. If columns will be physically stored in a bucket, then the total storage required will encompass all columns in the minhash matrix. Thus, the total storage required will remain at 4*m* KB. If, instead, the columns just keep track of how many columns are similar enough to hash to them, and do not store the physical columns, then the total storage required is $\frac{4m}{k}$ KB.

Question 3

1. **TF.IDF** is an acronym which stands for Term Frequency-Inverse Document Frequency. **TF.IDF** is a numerical statistic that represents the "importance" (or "relevance") of a word within a document or corpus. Unsurprisingly, **TF.IDF** was first created to aid in document search and information retrieval and is very useful for applications regarding text analysis and natural language processing.

2. In order to calculate the **TF.IDF** of a word, simply calculate the term frequency (i.e. TF) of that word in a specific document and multiply that by the word's inverse document frequency (i.e. IDF). The term frequency of a word is the number of times that word appears in a given document; it is possible to represent a word's "frequency" differently so long as this is consistent across all words. The inverse document frequency is the logarithm of the quotient of the total number of documents divided by the number of documents that contain the word. Multiplying these two values results in the **TF.IDF** of a word in a specific document. [1]

   Thus, the **TF.IDF** of the words *petite* and *cute* can be calculated as follows:

   $$TF.IDF(petite,\ d,\ D)\ = freq(petite,\ d)\ *\ log(\frac{count(petite,\ D)}{|D|})$$

   $$TF.IDF(cute,\ d,\ D)\ = freq(cute,\ d)\ *\ log(\frac{count(cute,\ D)}{|D|})$$

   where:

   d → a particular document
   D → the entire document set
   |D| → the cardinality (size) of the document set
   freq → a function that calculates the frequency of the passed word in the passed document
   count → a function that calculate the number of documents in the passed document set that contains the passed word

3. In the line `lda = LDA(k=7, seed=1, optimizer="em")`, the variable *k*, according to the PySpark documentation, signifies the number of soft cluster centers that should be inferred. The default value of *k* is 10 and *k* must be greater than one. [2]

[1] https://monkeylearn.com/blog/what-is-tf-idf/
[2] https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.clustering.LDA.html#pyspark.ml.clustering.LDA.k

4. Below are the words returned by `model_show()` for each topic and what I think each topic signifies.

    a. Topic #0: Clothing sizes

```
Topic #0:
        dress -> 0.028182701547527594
        small -> 0.027832747042053282
        large -> 0.02058385291461371
        runs -> 0.020129018880741312
        skirt -> 0.019860498600678154
        medium -> 0.019682849538786358
        waist -> 0.018200423219292328
        around -> 0.01426401480688629
        tight -> 0.014178872208616668
        beautiful -> 0.014013958311667992
```

    b. Topic #1: Describing a sweater

```
Topic #1:
        color -> 0.024985289320535856
        sweater -> 0.01936533961894519
        sleeves -> 0.0167663733279608
        pretty -> 0.015329011936476877
        blue -> 0.01450495313372339
        make -> 0.014398333886865598
        like -> 0.01396750434393409
        makes -> 0.013601189567140369
        looks -> 0.013077081301852157
        design -> 0.01240119369410706
```

    c. Topic #2: Describing a shirt

```
Topic #2:
        wear -> 0.034377182731960075
        bought -> 0.03074524227777064
        shirt -> 0.02233366830828211
        wearing -> 0.0193755957050799
        love -> 0.017248138427799167
        white -> 0.016984092609042597
        many -> 0.01639273557123052
        looking -> 0.015202556546059781
        wore -> 0.014677773607030329
        light -> 0.014556312795236859
```

d. Topic #3: Describing a a pair of jeans

```
Topic #3:
        great -> 0.03714535491863094
        perfect -> 0.02519506095368336
        soft -> 0.024499941369934178
        comfortable -> 0.02051995335584194
        super -> 0.02035514523322536
        jeans -> 0.01861059926460949
        black -> 0.01842669110431034
        pants -> 0.0173807699966845
        love -> 0.016272130187862513
        worn -> 0.015466732153873957
```

e. Topic #4: Describing a piece of clothing

```
Topic #4:
        petite -> 0.023947444685497643
        length -> 0.02223181022660492
        flattering -> 0.022122910250445327
        nice -> 0.018753611713761834
        fits -> 0.01767536959759522
        short -> 0.01766204251282486
        work -> 0.017216078077910413
        colors -> 0.015830450094419343
        general -> 0.015600142816671832
        little -> 0.013504614000422488
```

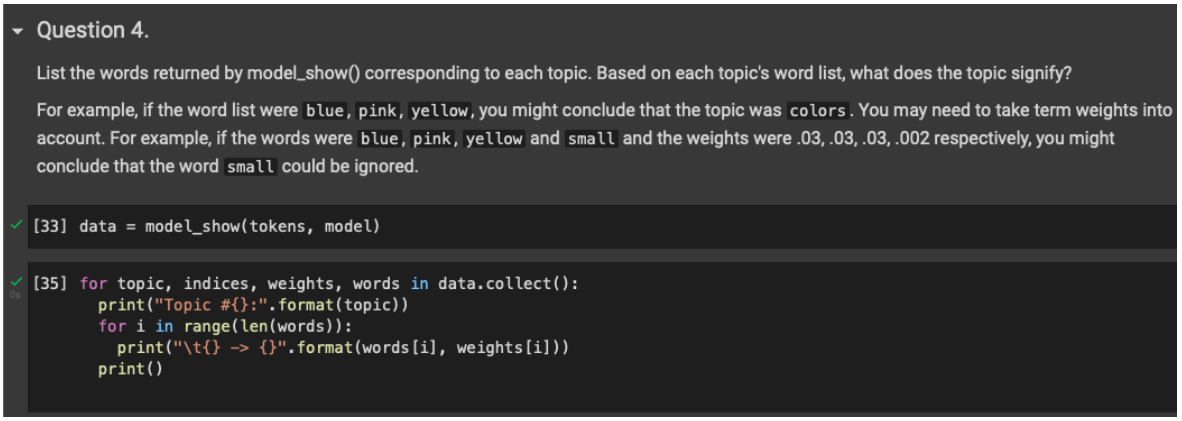
f. Topic #5: Describing a dress

```
Topic #5:
        really -> 0.019516091040072354
        quality -> 0.019362628209692395
        dress -> 0.01915935999126811
        good -> 0.019149757897377658
        made -> 0.018741913815260604
        like -> 0.017636018255578752
        material -> 0.017429170690941624
        even -> 0.017264161695697413
        going -> 0.015156829858148523
        front -> 0.014353025575156055
```

g. Topic #6: Shopping

```
Topic #6:
        size -> 0.0302811017675041
        ordered -> 0.029364043630358286
        would -> 0.02407341342662613
        tried -> 0.023818972195696085
        store -> 0.018090412303822936
        retailer -> 0.017602107713293225
        looked -> 0.017401743266185727
        loved -> 0.017028586014071218
        thought -> 0.015246147510015393
        online -> 0.014481857760306794
```

Notebook Code



The Colab Notebook can be found here.