Data Science Lab
Lab #3

Politecnico di Torino
October 23-24, 2019

Intro

In this laboratory, you will learn more about frequent itemsets and association rules. You will first use an existing library that implements Apriori and FP-Growth. Then, you will implement your own version of Apriori.

1 Preliminary steps

1.1 mlxtend and pandas

Make sure you have these two libraries installed. You can check whether a given library is installed by importing it in Python.

```
import mlxtend
import pandas
```

Or, if you use it, by checking `pip`:

```
$ pip freeze | egrep "(mlxtend|pandas)"
mlxtend==0.17.0
pandas==0.25.1
```

If not available, you will need to install them with `pip install [package]` (or any other package manager you may be using).

Pandas is a library for easy-to-use data structures and data analysis tools in Python. You will be learning about this library later in this course but, for the time being, you will use some basic functionalities, as it is required to use Mlxtend.

Mlxtend (machine learning extensions) is a Python library of useful tools for the day-to-day data science tasks. For most of the course we will be using scikit-learn. However, Mlxtend implements various frequent itemsets algorithms that are missing from scikit-learn. As such, for this laboratory, we will be using it. You can find out more about Mlxtend on the official website.

1.2 Datasets

For this lab, two different datasets will be used. Here, you will learn more about them and how to retrieve them.

1.2.1 Online Retail Data Set

The Online Retail Data Set [1] is a dataset made available on the UCI Machine Learning repository. It which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based online retail.

The original version of the dataset is available on UCI ML as a .xlsx. For your convenience, we are also making it available as a CSV file at the following URL.

```
https://github.com/dbdmg/data-science-lab/raw/master/datasets/online_retail.csv
```

Each of the 541,909 rows contains an item that has been purchased by someone. Items can be grouped into invoices (you can think of these as receipts), where each invoice has been issued for a specific buyer, and can contain multiple items.

The columns contained in the CSV file are the following:

- InvoiceNo: Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter §c', it indicates a cancellation.
- StockCode: Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.
- Description: Product (item) name. Nominal.
- Quantity: The quantities of each product (item) per transaction. Numeric.
- InvoiceDate: Invice Date and time. Numeric, the day and time when each transaction was generated.
- UnitPrice: Unit price. Numeric, Product price per unit in sterling.
- CustomerID: Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.
- Country: Country name. Nominal, the name of the country where each customer resides.

As an example, the following lines all refer to the same invoice (`InvoiceNo = 574021`).

```
434006,574021,23301,GARDENERS KNEELING PAD KEEP CALM ,48,2011-11-02 12:18:00,1.45,14434.0,United Kingdom
434007,574021,23355,HOT WATER BOTTLE KEEP CALM,24,2011-11-02 12:18:00,4.15,14434.0,United Kingdom
434008,574021,23284,DOORMAT KEEP CALM AND COME IN,20,2011-11-02 12:18:00,7.08,14434.0,United Kingdom
```

1.2.2 COCO Dataset

COCO Dataset is a large-scale object detection, segmentation, and captioning dataset. It offers a large number of images (from various contexts) with annotations (i.e. structured information on the contents of the image). These annotations, in particular, regard the contents of the image and, in particular, the objects contained within.

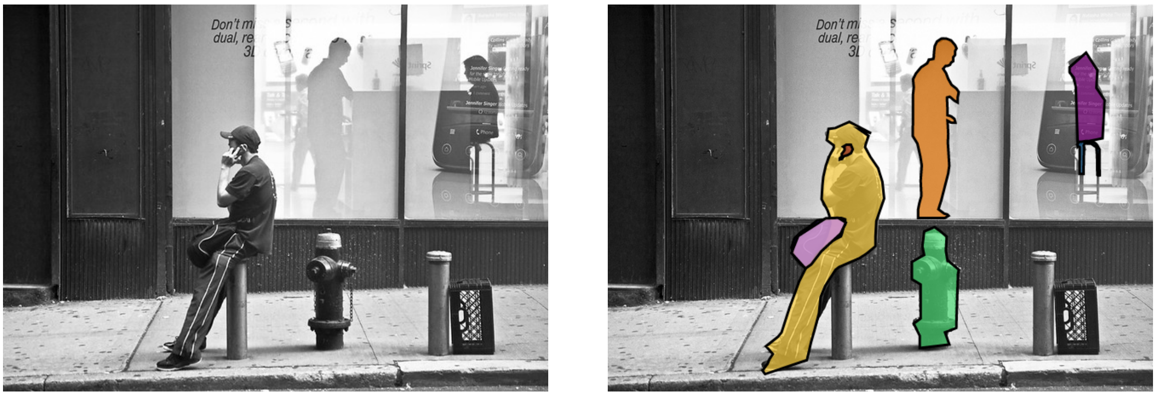
You can download a filtered and preprocessed version of COCO (which we will refer to as "modified") from the following URL:

```
https://raw.githubusercontent.com/dbdmg/data-science-lab/master/datasets/modified_coco.json
```

This dataset is a JSON file. You can open it using the already introduced `json` module. The file contains a list of images and, for each image, the `annotation` key contains all the annotations available. The following are the annotations for one such image.

```
{
    "file_name": "000000465265.png",
    "image_id": 465265,
    "annotations": [
        "person",
        "person",
        "person",
        "fire hydrant",
        "handbag",
        "chair",
        "cell phone"
    ]
}
```

This means that the image contains 3 people, a fire hydrant, a handbag, a chair and a cell phone. Indeed the actual image is represented in Figure 1 (with and without annotations visualized (also known as "segments")). You can can access each image (identified by the `image_id`) on the website, at: http://cocodataset.org/explore.



(a) Not segmented

(b) Segmented

Figure 1: Image #465265, with and without segmentation

2 Exercises

Note that exercises marked with a (*) are optional, you should focus on completing the other ones first.

2.1 Association rules from frequent itemsets

This exercise will work on the Online Retail Data Set. In particular, you will do some data preprocessing on the dataset to extract all itemsets available (where each itemset is the collection of items contained in a single invoice). Then, using FP-Growth and Apriori implementations, you will extract a list of frequent itemsets. From those, you will finally extract several different association rules.

1. First, you need to load the dataset into memory, using the `csv` module. Make sure you identify all valid rows. Also consider that rows having an InvoiceNo that starts with `C` should be discarded, as they indicate that the invoice is about a cancelled purchase.

2. Now that you have a dataset of items, you should aggregate it at an "invoice" level. For each invoice (identified by InvoiceNo) there can be multiple items (from multiple rows) in the dataset. For each invoice, you should build a list of all items belonging to it. For the example invoice presented in 1.2.1, you want to build the following list:

   how to know if there are wrong entries?

   ```
   [ "GARDENERS KNEELING PAD KEEP CALM",
     "HOT WATER BOTTLE KEEP CALM",
     "DOORMAT KEEP CALM AND COME IN" ]
   ```

3. You should now have a list (one for each invoice) of lists (each list containing the items bought for that invoice). Now, we need to convert this into a matrix form. Of the many possible formats, we will use the one expected by the Mlxtend library, which is as follows. Given an ordered list of M possible items (in this case, all possible products that can be bought), and given N itemsets (in this case, invoices), we should build a matrix of N rows and M columns. The element at the i^{th} row and j^{th} column should be 1 if the i^{th} itemset (invoice) contains the j^{th} item (product), 0 otherwise. For the following example:

   ```
   a,b,c
   b,c
   a,c,d
   a,b
   ```

   The list of all possible items is `[a, b, c, d]`. As such, the matrix that we will build is the following:

```
1 1 1 0
0 1 1 0
1 0 1 1
1 1 0 0
```

Once we have defined this matrix (as a list of lists), we can use Pandas to convert it to a *DataFrame* (which is, essentially, a table) with the following code:

```
import pandas as pd

all_items = ['a', 'b', 'c', 'd'] # this is your list of items
pa_matrix = [
    [1,1,1,0],
    [0,1,1,0],
    [1,0,1,1],
    [1,1,0,0]
] # this is the matrix you built from the itemsets
df = pd.DataFrame(data=pa_matrix, columns=all_items)
```

4. With the `df` that you defined in the previous exercise, you can now use the `fp_growth` function. This function, which is described in the detail in the official documentation. The first argument required is the previously built DataFrame, `df`. The second is the minimum support (*minsup*), i.e. the minimum fraction of the entire dataset in which the itemset should show up for it to be considered “frequent”. Try using different values of *minsup*, such as 0.5, 0.1, 0.05, 0.02, 0.01. How many results do you obtain as *minsup* varies? You can check the number of frequent itemsets identified and print them all with the following code snipped:

   ```
   fi = fpgrowth(df, 0.05)
   print(len(fi))
   print(fi.to_string())
   ```

5. Consider the itemsets extracted for *minsup* = 0.02. How many items are contained? Which ones would you be considered the most useful?

6. Use the value returned by `fpgrowth` to extract the relevant association rules.

7. Extract the association rules from the frequent itemsets extracted with *minsup* = 0.01. You can find the documentation for `association_rules()` on the official documentation. You can use the confidence as the metric to identify the rules, and a minimum threshold of 0.85 (feel free to vary these values and observe how the results vary).

8. (*) Rerun the experiments from point 4 with `apriori()` (documentation on the official website). Do the results match with the ones found by FP-Growth? Is Apriori faster or slower than FP-Growth? You can measure how long a function call takes with the following code snippet:

   ```
   import timeit

   # number=1 means that it executes the function only once
   timeit.timeit(lambda: apriori(df, 0.01), number=1)
   ```

2.2 Apriori implementation

In this exercise, you will implement your own version of Apriori and use it on the COCO dataset to extract frequent itemsets (i.e. groups of annotations that often co-occur within the same image, such as 'car' and 'traffic light').

Note that, while this entire exercise is optional, we recommend you try to solve it anyway. You may argue that there are libraries already implementing these and other algorithms. Despite that, we believe it is important, for a data scientist, to know the underlying theory as well as some implementation details. You can learn the former on textbooks, but you will only be faced with the latter when actually working on the implementation.

1. You can find a thorough explanation of the Apriori algorithm on the course slides on association rules (slides 16-20). With these, implement your own version of Apriori. You can use the below toy dataset from the example in slides 21-35 for initial troubleshooting and testing.

   ```
   a,b
   b,c,d
   a,c,d,e
   a,d,e
   a,b,c
   a,b,c,d
   b,c
   a,b,c
   a,b,d
   b,c,e
   ```

   When run with *minsup* > 1 (or 0.1 in relative terms), the expected itemset (with their *minsups*) are:

   ```
   a -> 7
   b -> 8
   c -> 7
   d -> 5
   e -> 3
   a,b -> 5
   a,c -> 4
   a,d -> 4
   a,e -> 2
   b,c -> 6
   b,d -> 3
   c,d -> 3
   c,e -> 2
   d,e -> 2
   a,b,c -> 3
   a,b,d -> 2
   a,c,d -> 2
   a,d,e -> 2
   b,c,d -> 2
   ```

2. Once you have implemented a working version of Apriori, you can load the modified COCO dataset from Subsection 1.2.2 into memory. From this, you should transform the dataset into a version compatible with the expected input of your Apriori implementation.

3. You can now run your implementation on the modified COCO dataset. Try using a *minsup* of 0.02, as well as other values. Are the obtained results meaningful? You can use the COCO dataset "explore" tool to examine any of the image used.

4. Now convert the modified COCO dataset into the format required by Mlxtend's `apriori()` and `fpgrowth()` functions (i.e. the one described in Exercise 3). Then, run these two algorithms on this dataset. Do the results from these functions match your results? (Hint: they should!)

5. How do apriori() and fpgrowth() compare in terms of execution time with your implementation? You can use the previously introduced timeit library to run comparisons.

6. (*) Finally, you can try with an exhaustive approach to finding frequent itemsets. This approach consists in generating all possible itemsets of length $\leq k$ and, for each one, counting the number of occurrences within the dataset. Try running it with low values of k (start from 1). How long does that take? How does the number of possible itemsets grow with k? Would that be feasible?

References

[1] D. Chen, S. L. Sain, and K. Guo, "Data mining for the online retail industry: A case study of rfm model-based customer segmentation using data mining," *Journal of Database Marketing & Customer Strategy Management*, vol. 19, no. 3, pp. 197–208, 2012.