[Date]

**Aleksandra Petkova - S6222041**

**Nour Aldin**

**Victor Essien**

Data Mining

**Case Study Report**

Contents

[Abstract 2](#_Toc36676572)

[Introduction 2](#_Toc36676573)

[Datasets 2](#_Toc36676574)

[1. Online Retail: time series, transactional data – 8 attributes, 541K records 2](#_Toc36676575)

[2. Groceries: customer recite data – 2](#_Toc36676576)

[3. MovieLens: movie rating data – 2](#_Toc36676577)

[Techniques 2](#_Toc36676578)

[*Association Rules Mining, Apriori – Aleksandra Petkova, Nour Aldin* 2](#_Toc36676579)

[Brief 2](#_Toc36676580)

[Datasets Usage 3](#_Toc36676581)

[Algorithm (*Source code in AssociationRulesCore.py*) 4](#_Toc36676582)

[Questions 4](#_Toc36676583)

[Experiments and Evaluation (What experiments can be done to prove it works) 5](#_Toc36676584)

[*Collaborative Filtering (Library) – Victor Essien* 5](#_Toc36676585)

[Brief 5](#_Toc36676586)

[Datasets 5](#_Toc36676587)

[Algorithm 5](#_Toc36676588)

[Experiments and Evaluation (What experiments can be done to prove it works) 5](#_Toc36676589)

[Conclusion 5](#_Toc36676590)

[References 5](#_Toc36676591)

**PLEASE DO NOT REMOVE COMENTS UNTIL THE FINAL DRAFT IS COMPLETE!**

# Abstract

“““Talk about..

Given data:  
Important Parameters:   
Analysis techniques used:   
Results/Output/Evaluation: ”””

# ABSTRACT HERE

# Introduction

# Intro paragraph: What is this about and what are we trying to do.

This project was developed for the Data Mining module at Teesside University with the aim to demonstrate and evaluate the use of popular computational techniques for data mining. The project contains implementations of Association Rules Mining, Collaborative Filtering and variety of datasets to test on.

# Datasets

### [Online Retail](https://archive.ics.uci.edu/ml/datasets/Online+Retail): time series, transactional data – 8 attributes, 541K records

**Each row represents a *whole sale purchase* of an item along with details about the purchase such as: the *invoice ID*, *stock code*, item *description*, *quantity* purchased, *price per unit*, the *ID* and *country* of the *customer* who made the purchase.**

### [Groceries](https://www.kaggle.com/irfanasrullah/groceries): customer recite data –

# More details here

### [MovieLens](https://www.kaggle.com/prajitdatta/movielens-100k-dataset): movie rating data –

# More details here

# Techniques

## *Association Rules Mining, Apriori – Aleksandra Petkova, Nour Aldin*

### Brief

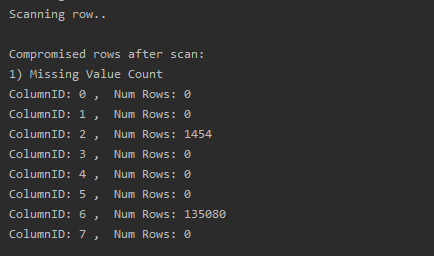
The **Apriori Algorithm** is one of many algorithms for **Association Rules Mining**, other popular examples are the **Eclat Algorithm**, **OPUS Search** and **FP-growth algorithm**. We chose to work with Apriori because … and it is also the one we are most familiar with.

* What is this technique normally used for?
* What other techniques exist for this problem?
* Why is this technique suitable for the selected dataset/ Why is this technique suitable for the selected dataset?

### Datasets Usage

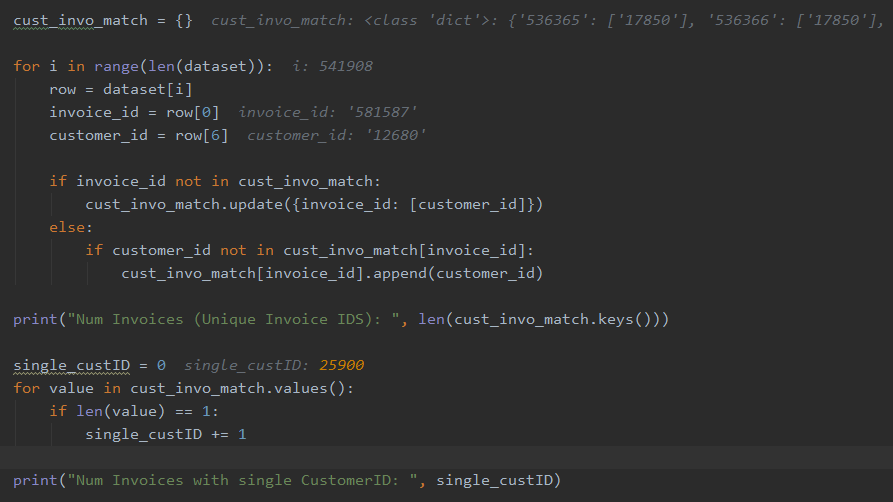
1. *Online Retail –* Used for both **the algorithm** **written from scratch** and **the algorithm** **written** **with the MLextend Library**. It is the **largest dataset** from the available and has both **missing** and **erroneous** **data**, those are the key reasons it was the dataset chosen to present the **Apriori Association Rules Mining Algorithm**.
2. *Groceries* – Much smaller dataset, no cleaning required, very simple to work with when building an **Association Rules Mining Algorithm**. That is why this is the dataset used to build and test the first prototype of the algorithm. (Not compatible with the current version MLextend Library version of the algorithm.)
3. *Simple Dataset* – Is a random **sample** from the *Groceries* dataset used to **validate** each step of the algorithm.

**Pre-processing Done and Experiments**:

1. **Missing Data Scan**: As you can see from Figure 1, there are only 2 columns that contain missing data – **Column 2 (Description)** and **Column 6 (CustomerID)**. The rows missing in **Column 6** representabout a **5th** of the whole dataset so ideally it would be good to fill them. According to the dataset structure (more information in the [Datasets Section](#_Online_Retail:_time)) the invoice is always sent to the same customer, i.e. the **InvoiceID** will only ever be linked to one **CustomerID**. This is confirmed by the investigation on this and the results from the mining shown on Figure 2.

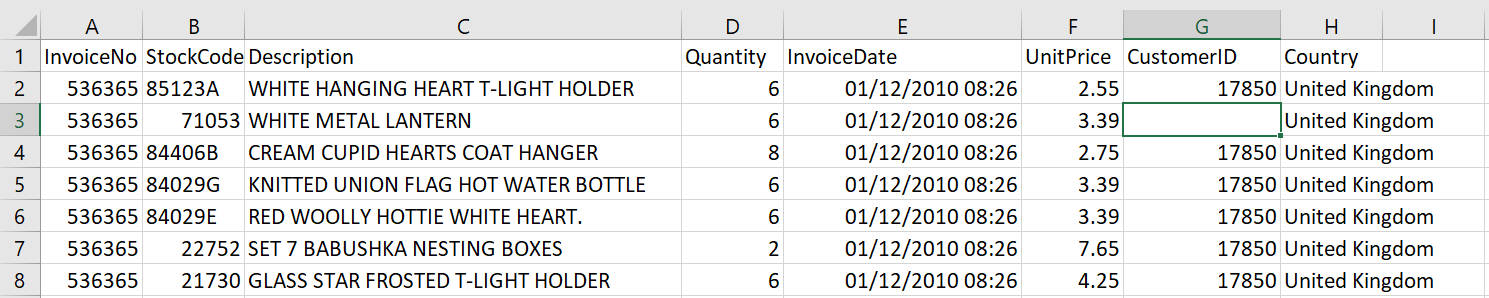
**Figure 1: Missing Data Scan**

**Output:**



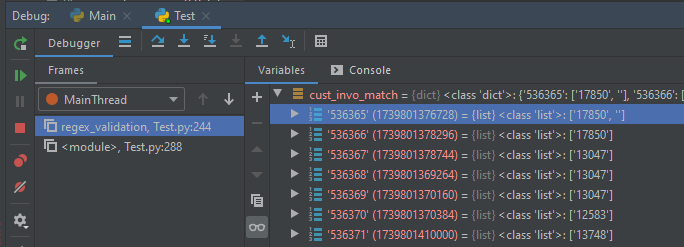
**Figure 2: Code written to prove theory, output below.**

TODO: Remove all the missing data then scan the dataset with the regex validation and add a new image for figure 5. This can potentially be done by taking the missing data scan out of the regex\_validation function and calling the function right after the missing data scan and removal.

If there was a case where one or more **CustomerIDs** existed in an Invoice with missing **CustomerIDs** that would mean the **cust\_invo\_match dictionary** would store the existing **CustomerID** and the empty string - **“”**. In the example shown below (Figure 3) I manually removed one of the customer IDs from an invoice to test how the scan will react to this.

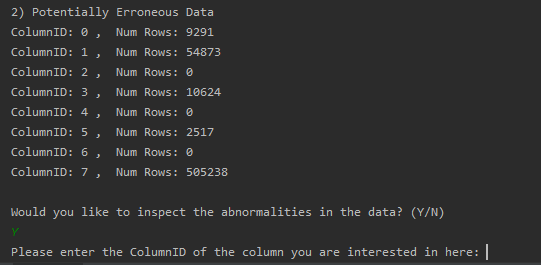
**Figure 3, Experiment: Manually removing a CustomerID.**

The empty string was indeed added as an option for **CustomerID** for the current **InvoiceID** (Figure 4). The fact that every invoice in the dataset only has one option for **CustomerID**, as shown on Figure 2 – output, means that there are no cases where at least one **CustomerID** is present on an invoice with one or more missing **Customer IDs**. Which means the idea to fill the data in this way is not possible.



**Figure 4: Experiment results after removing a customer ID.**

I could not find any other ways to fill the missing data so all of it was removed including the records with missing data in column 2.

1. **Built a Regular Expression Validation System:** Catches any data that does not fit the set rules for the format of a column. **Example Regex**: **UnitPrice column (ID: 5)** has the regular expression **“^£{0,1}([0-9]+\.[0-9]\*[1-9]+)|([1-9][0-9]\*)”** which states, the data in the column may have 0 or 1 of £ symbol at the start of the column data – **“^£{0,1}”**, followed by either a floating point number that is > 0.0 – **“[0-9]+\.[0-9]\*[1-9]+”**, or a whole number that is > 0 – **“[1-9][0-9]\*”.** If the data in the **column fails** this **validation**, it will be recorded into the **compromised\_rows** **dictionary**, which the user has the option to inspect right after the full scan is complete. The output after the scan is shown in the screenshot on Figure 5. If the user types Y or y, they will be asked to **select a column** by   
   entering the **column ID**.

**Figure 5: Regex Validation System Output**

### Algorithm (*Source code in AssociationRulesCore.py*)

1. Find all **unique items** by transforming the loaded dataset from 2D to 1D and applying a set cast on the whole dataset to remove any repetitions. The unique items are then sorted to allow for easier management of the data.
2. Create an item to number **map/dictionary**, with the itemset as the key and the item ID as the value to **improve performance** as string comparisons are expensive.
3. **Generate mapped data**
   1. Reformat the **unique items** using the **unique items map** so that each **1-itemsets** is represented by its **ID**.
   2. Reformat the **dataset** using the **unique items map** so that the algorithm is only working with item **IDs**.
4. Count the **frequency** of each unique item.
5. Calculate the **minimum support** by multiplying the **minimum relative support** by the highest **frequency**. (**Example**: if the highest frequency is 6 and the relative support is 0.5 (50%) the minimum support/frequency will be 3.)
6. Apply the **Apriori property** to filter out the **infrequent** 1-itemsets.
7. Append the survived 1-itemsets dictionary to the complete\_survival\_subsets\_list.
8. Start main Association Rules Loop and keep running until there are no more itemsets left to work with.

8.1. Generate a k+1 itemsets list from the survived k-itemsets dictionary.

8.1.1 Sets contain only unique values so {0, 0}, {1,1}, {2,2}, etc should not exist! Also, multiple variations of the same subset/itemset should not be checked at this stage because they may not and most likely will not be frequent. Example: if {0,1} is checked {1,0} should not be checked unless for sure {0,1} is frequent. Then we want to see which item implies the other so we will try each variation. Below there are tables of the acceptable 2, 3, 4-itemsets that implement the rules mentioned. You can see that there is a clear pattern which was used to optimise the process for generating k+1 itemsets.

based on a much smaller dataset (**simple\_dataset.csv**) that is also available in the “**Data Repository**” folder. The example tables do not show the Apriori property, they simply visualise the outputs assuming all subsets survive.

Table :  **All possible set combinations 1-itemset 🡪 2-itemset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | {0} | {1} | {2} | {3} | {4} |
| {0} | X | X | X | X | X |
| {1} | {0, 1} | X | X | X | X |
| {2} | {0, 2} | {1, 2} | X | X | X |
| {3} | {0, 3} | {1, 3} | {2, 3} | X | X |
| {4} | {0, 4} | {1, 4} | {2, 4} | {3, 4} | X |

Table 2: **All possible set combinations** **2-itemset 🡪 3-itemset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | {0} | {1} | {2} | {3} | {4} |
| {0, 1} | X | X | X | X | X |
| {0, 2} | X | X | X | X | X |
| {0, 3} | X | X | X | X | X |
| {0, 4} | X | X | X | X | X |
| {1, 2} | {0,1,2} | X | X | X | X |
| {1, 3} | {0,1,3} | X | X | X | X |
| {1, 4} | {0,1,4} | X | X | X | X |
| {2, 3} | {0,2,3} | {1,2,3} | X | X | X |
| {2, 4} | {0,2,4} | {1,2,4} | X | X | X |
| {3, 4} | {0,3,4} | {1,3,4} | {2,3,4} | X | X |

Table : **All possible set combinations 3-itemset 🡪 4-itemset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | {0} | {1} | {2} | {3} | {4} |
| {0,1,2} | X | X | X | X | X |
| {0,1,3} | X | X | X | X | X |
| {0,1,4} | X | X | X | X | X |
| {0,2,3} | X | X | X | X | X |
| {0,2,4} | X | X | X | X | X |
| {0,3,4} | X | X | X | X | X |
| {1,2,3} | {0,1,2,3} | X | X | X | X |
| {1,2,4} | {0,1,2,4} | X | X | X | X |
| {1,3,4} | {0,1,3,4} | X | X | X | X |
| {2,3,4} | {0,2,3,4} | {1,2,3,4} | X | X | X |

8.2. Count the **frequency** of each itemset.

8.3. Apply the **Apriori property** to filter out the **infrequent** current k-itemsets.

1. Generate Rules out of the frequent itemsets.

9.1. Ignore the 1-itemsets as no rules can be generated from 1 item.

9.2. Split the itemset into two parts (X and Y).

9.3. Calculate **confidence** of the rule **X implies Y or Y implies X**.

9.4. If it is below the **minimum confidence** given, discard the rule.

1. Display rules

10.1. Open the file where the rules will be written - **association\_rules.txt** (the “with” keyword was used to make sure Python’s garbage collector cleans the file object)

10.2. Go through every rule in the rules dictionary

10.2.1. Extract the X and Y subsets, their frequencies/supports and the confidence of the rule.

10.2.2. Construct a string out of that data to display the results.

**Example string:** **{pip fruit, root vegetables}**, **sup(153), rel sup(2%) ---> {whole milk}, sup(2513), rel sup(26%) - conf(58%)**

1. Count the number of rules and itemsets and print that to the console.

### Questions

1. How many **unique items** are there?
2. What is the **frequency** of each **unique item**?
3. Which items have the **highest frequency**?
4. Which items are very **often bought together**? (Itemsets with **relative support higher than 50%**)
5. What **association rules** are there about the items in the dataset/s?
6. Are there any **associations** in the dataset/s with **high confidence**? (**Rule confidence > 50%**)
7. When given an **itemset** can the algorithm **return all it’s association rules**?

### Experiments and Evaluation (What experiments can be done to prove it works)

1. Display results with different support values.
2. Checked if the values make sense.
3. Applied algorithm on a very small dataset (**simple\_dataset.csv**) so that results can easily be verified.

## *Collaborative Filtering (Library) – Victor Essien*

### Brief

### Datasets

1. *MovieLens*

### Algorithm

### Experiments and Evaluation (What experiments can be done to prove it works)

1. Q1
2. Q2

# Conclusion

# References

**There are no sources in the current document.**