DATA MINING PROJECT

Pierre Arbelet, Arthur Dupont, Anthony Hayot

December 11, 2014

1 Introduction

The product offer of supermarkets is large and complex and it can be very challenging for consumers to navigate into it. This complexity is an obstacle for conducting basic actions such as:

- comparing price between supermarkets
- replace products to fit in a specific budget
- switch products to match a specific behaviour (organic, hallal, gluten-free...)

The objective of this project is to provide the first brick, which is a similarity metric between products, of an efficient shopping experience.

The following steps have been followed:

- Step 1: Crawl product inventories from online supermarkets
- Step 2: Data cleaning (avoid redundancy, completion, formating)
- Step 3: Build Distance Functions between products to allow efficient substitutions

2 Crawling inventories

For this project, we have focused our efforts on 3 online supermakets: Auchan, Monoprix and Simply. But the approach that we will detail below can be easily applied to any supermarket website, as they all share a similar architecture:

- a welcome page with links to different categories (beverage, meat, fruit...)
- inside each category, different levels of sub-categories (alcohol, wine, red wine)
- after a couple of categorization levels (usually between 1 and 5), a list of products
- by clicking on each product, "product pages " where detailed information (description, energetic values, price...) is displayed

Our crawlers are built in two steps:

- navigate through the categories to collect "product page" urls
- extract relevant information for each "product page"

The Python code for the first step is described below:

```
In []: import requests
       from bs4 import BeautifulSoup
       import pandas as pd
       import re

       #different levels of categorization
       rayonsA=[]
       rayonsB=[]
       rayonsC=[]
       rayonsD=[]
       rayonsE=[]
       productLink=[]

       #start at welcome page and get first level categories
       url='http://www.livraison.simplymarket.fr/'
       r = requests.get(url)
       soup = BeautifulSoup(r.text,'html.parser')
       balises_a=soup.find_all("a",class_='linkMenu')

       for balise_a in balises_a:
               rayonsA.append('http://www.livraison.simplymarket.fr/'+balise_a.get('href'))

       #navigate in each Level-1 category
       for rayonA in rayonsA:
               r = requests.get(rayonA)
               soup = BeautifulSoup(r.text,'html.parser')
               balises_a=soup.find_all("a",class_='lienProduit')
               balises_tr=soup.find_all("tr",class_='trLibelle')

       #find product pages, if any, and add it to the list
               for balise_a in balises_a:
                       productLink.append('http://www.livraison.simplymarket.fr/'+balise_a.get('href'))

       #if no product pages can be found, find Level-2 categories
               if balises_a==[]:
                       for balise_tr in balises_tr:
                               rayonsB.append('http://www.livraison.simplymarket.fr/'+balise_tr.find("a
```

This algorithm is repeated over each level of categorization until no new category is discovered. At the end, the variable productLink is a list that contains all the "product page" urls and we can move to the second step.

The second step consists in scrapping the relevant part of the html code of each "product page" that contains the following fields:

- product_name
- brand
- quantity (eg 6 for 6x33cL)
- weight/volume (eg 33 for 6x33cL)
- weight/volume_total (eg 200 for 6x33cL)
- unit (eg cL)

- description
- ingredients
- preservation
- nutritional details
- origin
- price
- price per unit

3 Data Cleaning

Once the html code containing these data has been scrapped, it needs to be parsed and cleaned in order to fill the table with the correct values. Ideally, we would have stored the code in a text database, such as MongoDB, and then build a parser to transfer information from MongoDB to a standard relationnal database. For this project, we have scrapped and cleaned the data at the same time and built a csv file to store it.
First, the parsing/cleaning stage consists in understanding the structure of the html code. For example:

In []:

```
            <div class="redactionnel">
                <div class="texteProduit">
                    <h1>FOIE GRAS DE CANARD ENTIER SUD OUEST GASTRONOMIQUE MONTFORT 300G - Montfort</h1>
                </div>
                        <div class="texteProduit">
                    <label>Prix quantité</label><br>
                    99,67 eur/Kg
                </div>
                        <div class="texteProduit">
                    <label>Descriptif</label><br>
                    Foie gras de canard entier du Sud Ouest.
                </div>
                        <div class="texteProduit">
                    <label>Avantages</label><br>
                    <div>La recette du Gastronomique est inspirée des préparations des plus grands Chefs
                </div>
                        <div class="texteProduit">
                    <label>Ingrédients</label><br>
                    Foie gras de canard, sel, Armagnac, Porto, poivre, sucre, antioxydant : ascorbate de
                </div>
                        <div class="texteProduit">
                    <label>Conservation</label><br>
                    A conserver au réfrigérateur entre 0°C et +4°C. A consommer rapidement après ouvertu
                </div>
                        <div class="texteProduit">
                    <label>Renseignements pratiques</label><br>
                    <p>Sortir le produit du réfrigérateur quelques minutes avant de le déguster.</p><p>E
                </div>
                        <div class="texteProduit">
                <table width="100%" cellspacing="0" cellpadding="0">
                    <tbody><tr>
```

For this example, we can observe that each piece of information is preceded by its label (eg 'Conservation'). The parser will try to find these labels and if they exist, it will take the information that comes right after and store it in the relevant variable.

```
In []: if 'Conservation' in reduced_prod_info:
                                index_conservation=reduced_prod_info.index('Conservation')
                                conservation=reduced_prod_info[index_conservation+1]
```

Some data are stored in a different format than expected in our database. To solve this, we have used regular expressions in order to recognize patterns and extract exactly what we were looking for.
Example:
FOIE GRAS DE CANARD ENTIER SUD OUEST GASTRONOMIQUE MONTFORT 300G - Montfort

We want to have:

- name = FOIE GRAS DE CANARD ENTIER SUD OUEST GASTRONOMIQUE MONTFORT
- quantity = 1
- weight/volume = 300
- unit=G

In Python, this will translate:

```
In []: title=re.match(r'(.*)\s(\d+\.?\d+?)(KG|G|ML|CL|L)',nom)
                        if title!=None:
                                nom=title.group(1)
                                quantite=1
                                poids_volume_total=title.group(2)
                                unite=title.group(3)
```

By iterating this operation on most observed formats, we managed to build a clean dataset available for analysis.

4 Distance between products

Now that we have a clean dataset, we need to build a distance function between products in order to find, for each product, the most similar items in our inventory. To build this distance function, we started with distance between identical fields. This could be very simple for numeric fields such as weight/volume but is more complex for Strings. We will focus on distance between Strings for this project.

4.1 Jaccard Distance

Our first call for performing distance between sets of Strings is by using Jaccard Distance defined by:

Jaccard Distance $= \frac{card(A \cup B) - card(A \cap B)}{card(A \cup B)}$

where $card(A \cup B)$ represents the number of distinct words in String A and B
and $card(A \cap B)$ represents the number of common words in String A and B.

In Python, it translates as below:

```
In []: # compute the Jaccard distance between two sentences
       def DistJaccard(str1, str2):
           if str1 != '' and str2 != '':
```

```
            str1 = set(str1.split())
            str2 = set(str2.split())
            return 1.0 - float(len(str1 & str2)) / len(str1 | str2)
        else:
            return numpy.nan
```

This method gives us a first hint of the distance between our two strings but has two weaknesses:

- irrelevant words (article, punctuations...) should not be counted
- variations on the same word (singular/plural, conjugation) should be counted as common words

4.2 Stopwords

Those irrelevant words are known as "Stopwords" and should be removed before performing the Jaccard Distance. To do so, we use a list of French Stopwords from https://code.google.com/p/stop-words/. We search for every words of A in the list. If it is in the list, the word is removed.
For example, the sentence:
Assortiment de chocolats au lait fourres et de biscuits enrobes de chocolat au lait.
becomes:
assortiment chocolats lait fourres biscuits enrobes chocolat lait
In Python, we have:

```
In []: # functions to normaliza data (remove stopwords, ponctuation and numbers)
       with open('stopwords_fr.txt') as f:
           Stopwords = [r.rstrip() for r in f.readlines()]

       exclude = set(string.punctuation)
       def remove_ponctuation(s):
           return ''.join(ch for ch in s if ch not in exclude)

       def remove_stopwords(s):
           return ' '.join(word for word in s.split() if word not in set(Stopwords))

       def normalize_data(s):
           return remove_stopwords(remove_ponctuation(str(s))).upper()

       # keep only strings
       def toString(sentence):
           out = ''
           if str(sentence) != 'nan':
               for word in sentence.split():
                   if isinstance(word, basestring):
                       out += (" " + word)
           return out
```

4.3 Lemmatization

The idea here is to be able to count words with the same root (eg chocolat, chocolats et chocolatée) as common words. To do so, we extract the root using FrenchStemmer from nltk.stem.snowball library. We, then, replace the word by its root.
Our example becomes:
assorti chocolat lait fourr biscuit enrob chocolat lait

By removing Stopwords and lemmatizing, we get a significant improvement that we will present in the next sections.

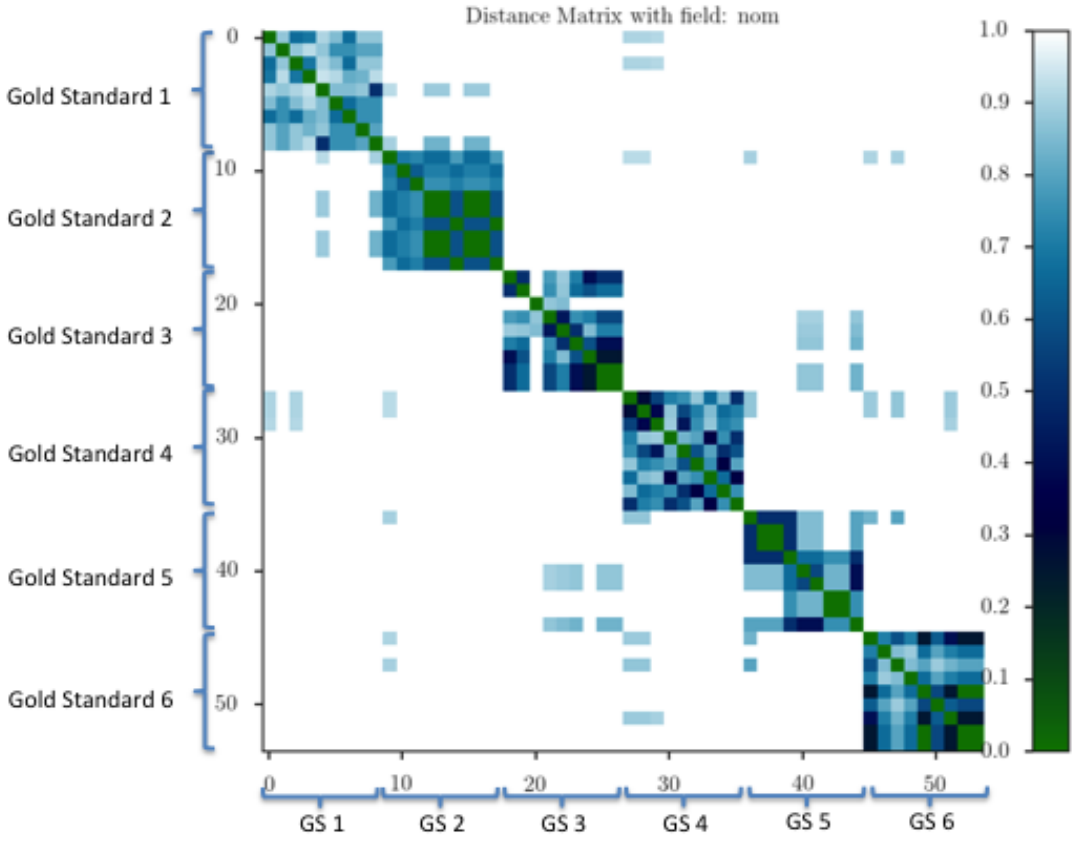
5 Results

To analyse the performance of our algorithm, we have manually define a "Gold Standard". We chose 6 initial products from the Simply inventory but in different product categories.
For each product, we picked:

- the 2 most similar products at Simply
- the 3 most similar products at Monoprix
- the 3 most similar products at Auchan

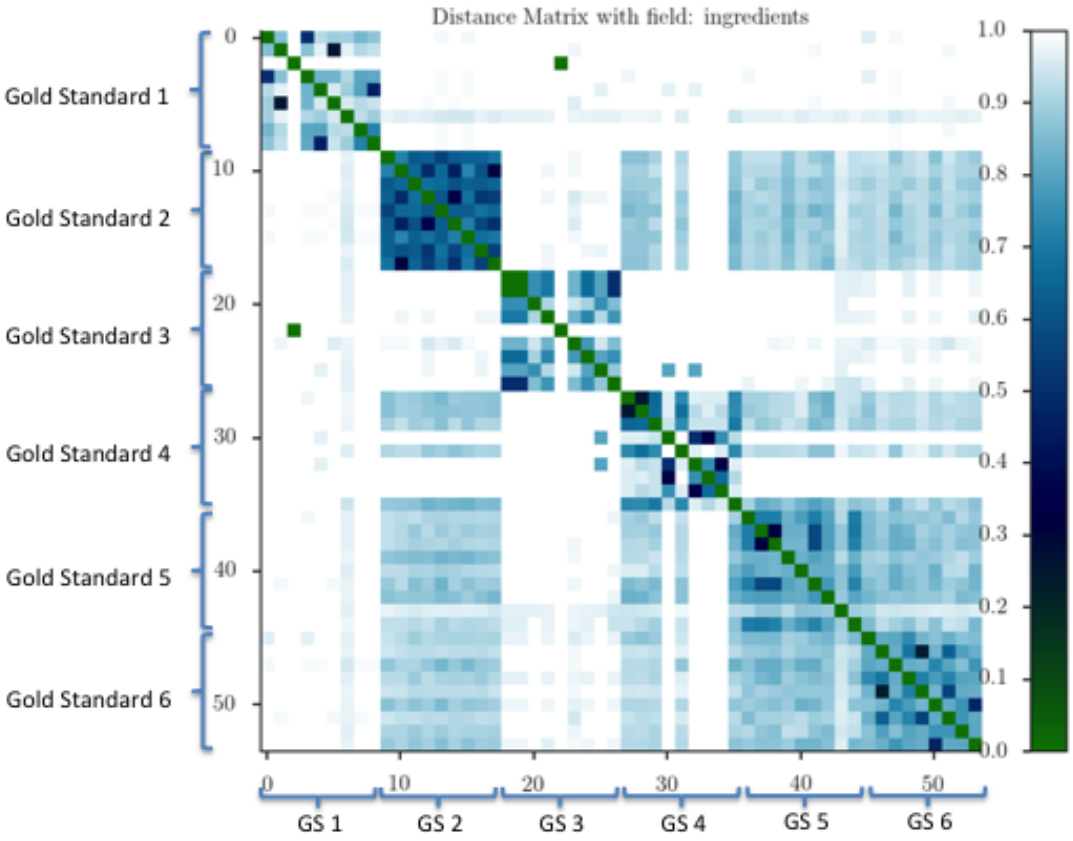
We call GoldStandard i the ensemble gathering the initial product i and its 8 similar products.

First, we compute the distance between each product of our Gold Standard for the fields "nom", " ingredient" and "description". The results are displayed below:

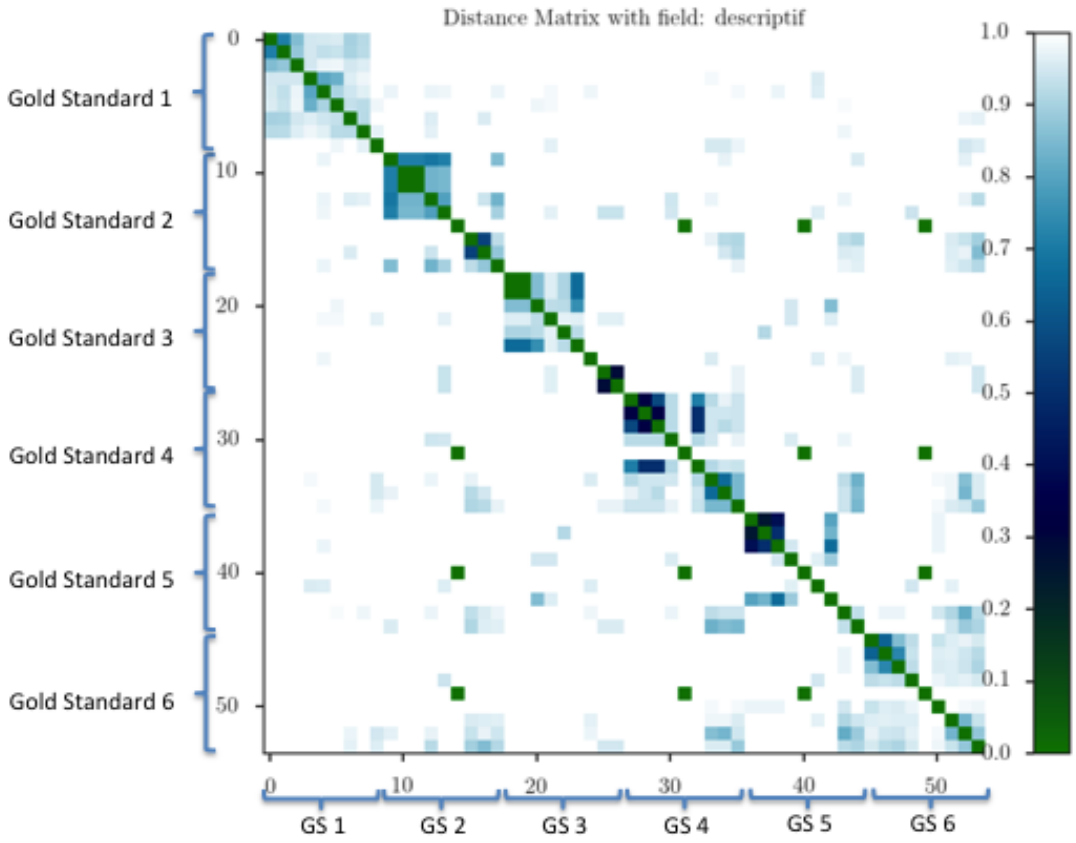


We can observe that our distance behaves well on that field. Inside a Gold Standard, the distance is very small between products. Outside, the distance is almost maximum.

On the field "ingredient", the results are less obvious. For some products, it seems to work fine while for some others, ingredients are less discriminant.



For the field "descriptif", it seems that it performs well inside a supermarket (3 products) because it uses a an homogeneous language but makes it less performant to compare products from different supermarkets.



Based on these results, we empirically chose the following weights :

- nom: 0,6
- ingredient: 0,2
- description: 0,2

Remark :
we computed the tf-idf on the ingredient field to select the 10 most important words in that field: the results were encouraging (see code in SimilarProducts.py).

We have then tested our algorithm (data normalization and weighted average of Jaccard distance) on the whole database for the 6 initial products. We obtained the results in Appendix. We will next compute a score comparing these results with the gold rule.

In order not to store the whole distance matrix (memory efficiency), we first store the 10 first distances and then, for each distance computed, if it is lower than the maximum distance in the list, we replace the maximum by this value.

6 Conclusion

This project was a very rich experience for us. We went through the different phases and had to find our way through technical difficulties especially in the crawling phase. Our first results seem encouraging and can consitute the first brick of a bigger project. The distance function can be improved by using more powerful semantic tools (LSA, LDA). We can also imagine to feed our algorithm with true shopper experiences in order to personnalize the substitution relevancy.

Type	Produit	Enseigne	Marque
Produit	PIZZA 4 FROMAGES AUCHAN	Simply	AUCHAN
Match	PIZZA 4 FROMAGES AUCHAN	Simply	AUCHAN
Match	Pizza aux 4 fromages	Auchan	BUITONI
Match	Pizza aux 4 fromages	Monoprix	Dr Oetker
Match	Pizza aux 4 fromages	Monoprix	Sodebo Dolce
Match	Pizza 4 fromages	Auchan	AUCHAN
Match	Pizza aux 4 fromages	Monoprix	Sodebo Presto
Match	PIZZA l'EXTRA MOELLEUSE 4 FROMAGES AUCHAN	Simply	AUCHAN
Match	PIZZA JAMBON FROMAGES AUCHAN	Simply	AUCHAN
Match	La pizza 4 fromages	Auchan	SODEBO
Produit	EPINARDS BRANCHES A LA CREME AUCHAN	Simply	AUCHAN
Match	EPINARDS BRANCHES A LA CREME AUCHAN	Simply	AUCHAN
Match	EPINARDS HACHES A LA CREME AUCHAN	Simply	AUCHAN
Match	EPINARDS HACHES A LA CREME FRAICHE FINDUS	Simply	Findus
Match	Epinards branches a la creme	Auchan	AUCHAN
Match	EPINARDS BRANCHES BIO AUCHAN	Simply	AUCHAN
Match	EPINARDS BRANCHES BIO AUCHAN	Simply	AUCHAN
Match	Epinards en branches a la creme fraiche	Auchan	FINDUS
Match	POISSON A LA BORDELAISE AUCHAN	Simply	AUCHAN
Match	BEIGNETS A LA ROMAINE AUCHAN	Simply	AUCHAN
Produit	LINGETTES BEBE A L'ALOE VERA POUCE	Simply	POUCE
Match	LINGETTES BEBE A L'ALOE VERA POUCE	Simply	POUCE
Match	Lingettes bebe	Auchan	POUCE
Match	Lingettes epaisses sans alcool pour bebe a leau nettoyante, a laloe vera	Monoprix	BoutChou
Match	Lingettes epaisses sans alcool pour bebe a leau nettoyante, a laloe vera	Monoprix	BoutChou
Match	Lingettes epaisses sans alcool pour bebe a leau nettoyante, a laloe vera	Monoprix	BoutChou
Match	Lingettes epaisses sans alcool pour bebe a leau nettoyante, a laloe vera	Monoprix	BoutChou
Match	Lingettes sensitives a laloe vera pour les peaux sensibles	Monoprix	BoutChou
Match	Lingettes sensitives a laloe vera pour les peaux sensibles	Monoprix	BoutChou
Match	Lingettes pour bebe	Auchan	PAMPERS
Produit	JUS D'ORANGE AVEC PULPE TROPICANA	Simply	Tropicana
Match	JUS D'ORANGE AVEC PULPE TROPICANA	Simply	Tropicana
Match	JUS D'ORANGE AVEC PULPE TROPICANA	Simply	Tropicana
Match	JUS D'ORANGE AVEC PULPE TROPICANA 1,75L	Simply	Tropicana
Match	JUS D'ORANGE AVEC PULPE TROPICANA 1,75L	Simply	Tropicana
Match	PUR JUS D'ORANGE AVEC PULPE TROPICANA	Simply	TROPICANA
Match	JUS D'ORANGE SANS PULPE TROPICANA	Simply	Tropicana
Match	JUS D'ORANGE SANS PULPE TROPICANA	Simply	Tropicana
Match	JUS D'ORANGE PULPISSIMO TROPICANA	Simply	Tropicana
Match	JUS D'ORANGE PULPISSIMO TROPICANA	Simply	Tropicana
Produit	RAVIOLI BOLOGNAISE AUCHAN	Simply	AUCHAN
Match	RAVIOLI BOLOGNAISE AUCHAN	Simply	AUCHAN
Match	RAVIOLI BOLOGNAISE ZAPETTI	Simply	ZAPETTI
Match	RAVIOLI BOLOGNAISE ZAPETTI	Simply	ZAPETTI
Match	SAUCE BOLOGNAISE AUCHAN	Simply	AUCHAN
Match	SAUCE BOLOGNAISE AUCHAN	Simply	AUCHAN
Match	RAVIOLI LEGUMES AUCHAN	Simply	AUCHAN
Match	RAVIOLI PUR BOEUF AUCHAN	Simply	AUCHAN
Match	RAVIOLI PUR BOEUF AUCHAN	Simply	AUCHAN
Match	LASAGNES BOLOGNAISE AUCHAN	Simply	AUCHAN

Type	Produit	Enseigne	Marque
Produit	PAIN AU LAIT AUX PEPITES DE CHOCOLAT AUCHAN	Simply	AUCHAN
Match	PAIN AU LAIT AUX PEPITES DE CHOCOLAT AUCHAN	Simply	AUCHAN
Match	PAIN AU LAIT AUCHAN	Simply	AUCHAN
Match	BISCUITS PETIT DEJEUNER PEPITES DE CHOCOLAT LAIT AUCHAN	Simply	AUCHAN
Match	DOO WAP PEPITES CHOCOLAT AU LAIT HARRY'S	Simply	HARRY'S
Match	PAIN CHOCOLAT AUCHAN	Simply	AUCHAN
Match	CHOCOLAT AU LAIT DEGUSTATION AUCHAN	Simply	AUCHAN
Match	BRIOCHES AUX PEPITES DE CHOCOLAT RIK&ROK;	Simply	RIK&ROK;
Match	PAIN AU LAIT PASQUIER	Simply	Pasquier
Match	Pepites Chocolat au lait	Monoprix	Vahine