Emilie JALRAS

Mise en place de systèmes de recommandations

Rapport de Stage

du 16 juin au 5 septembre
tuteur entreprise : Benoît Baccot
tuteur école : Vincent Charvillat

Remerciements

Avant de commencer ce rapport, j'aimerais remercier les personnes grâce à qui mon stage chez Devatics a été rendu possible.

Je remercie donc particulièrement Benoit Baccot, mon tuteur au sein de Devatics, qui s'est patiemment occupé de moi durant toute la durée de mon stage, ainsi que Vincent Charvillat et Sandrine Mouysset, professeurs et chercheurs à l'ENSEEIHT, qui m'ont permis de trouver ce stage, et m'ont suivie chaque semaine avec une motivation toujours présente.

Je remercie également Monsieur Romulus Grigoras, pour m'avoir accueillie au sein de son entreprise ainsi que les autres membres de l'équipe Devatics pour leur accueil chaleureux.

Sommaire

Introduction

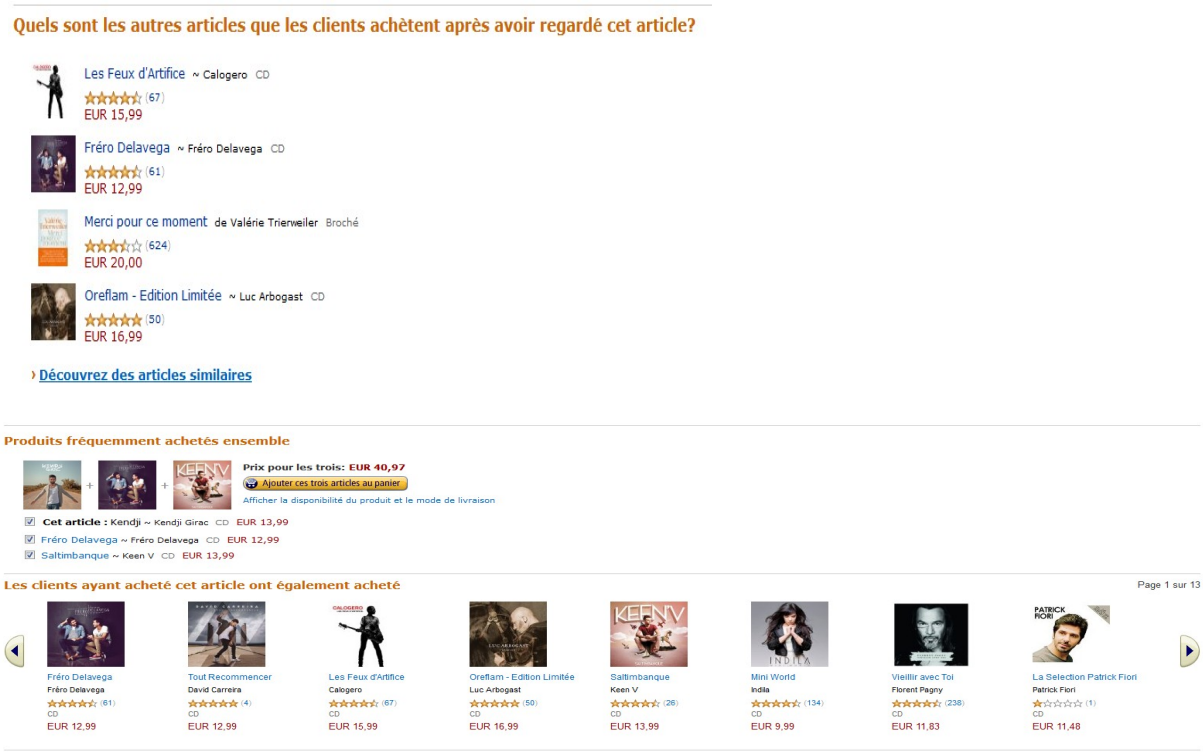
1. L'entreprise

Devatics est une entreprise basée à Toulouse, à côté de l'Enseeiht. Cette société propose une solution de commerce électronique qui se concentre sur la publicité et les technologies intelligentes d'analyse comportementale en temps réel. Fondée en 2010, elle est dotée d'une vingtaine de salariés, et est en pleine expansion. Elle travaille pour des clients tels que Kaporal Jeans, La Dépêche du Midi, TomPress, Teddy Smith et de nombreux autres.

2. Les systèmes de recommandations

La première chose que j'ai eu l'occasion de faire durant mon stage, a été de me renseigner sur les systèmes de recommandations existants.

Les systèmes de recommandations sont en plein essor, facilitant les ventes pour les sites d'e-commerce, ils ont été beaucoup développés par ces mêmes sites, par exemple Amazon :



Dans cet exemple en consultant la fiche produit d'un CD sur Amazon, on trouve en dessous de nombreuses recommandations, basées sur ce que les autres utilisateurs du site ont acheté. Ce sont donc des recommandations user-based.

On trouve en fait trois types de recommandations :

- user-based : ces recommandations sont basées sur les comportements des utilisateurs. (comme dans les exemples ci-dessus)
- content-based : ces recommandations sont basées uniquement sur les produits, par exemple en recommandant des produits similaires au produit étudié.
- mixed : c'est un mélange des deux méthodes précédentes : ces recommandations sont faites en utilisant à la fois des caractéristiques du produit, et une étude du comportement des utilisateurs.

Lors de mon stage je me suis concentrée uniquement sur les systèmes de recommandation basés sur le contenu.

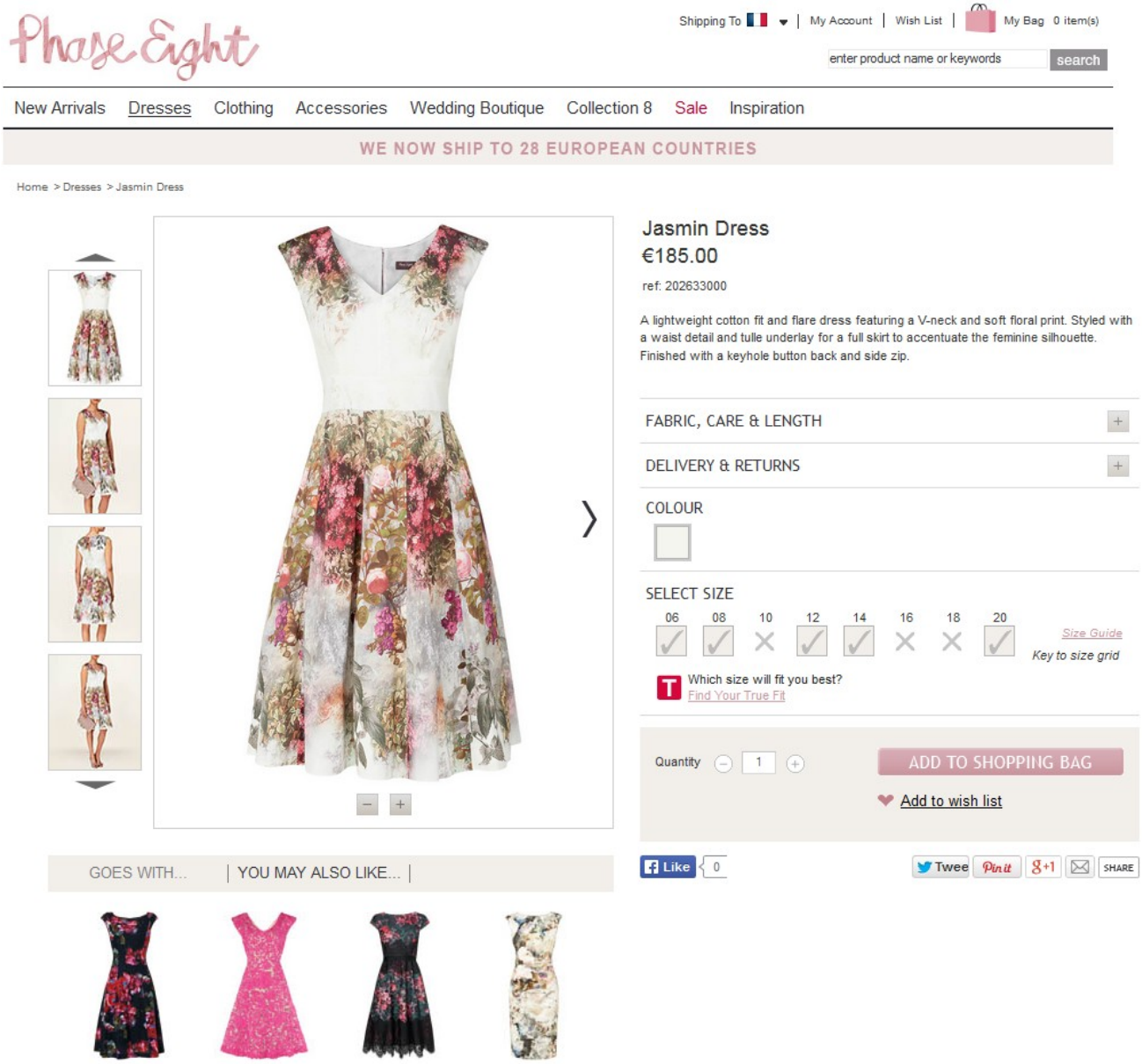
3. Objectif

L'objectif de mon stage était de réaliser des recommandations à partir de données réelles fournies par un des sites clients de Devatics : Phase Eight.

Phase Eight est une chaine de magasins de prêt à porter née en 1979 au Royaume-Uni et qui s'est étendue depuis à l'international (notamment en Europe, au Moyen-Orient ou encore en Asie).

En 2008 ils ont mis en place leur site internet, d'où sont effectuées des ventes vers une trentaine de pays différents. Ce site est visité par près de 800000 visiteurs chaque mois.

Actuellement voici la vue qu'a un acheteur lorsqu'il désire consulter des informations sur un produit.



On voit en bas une catégorie « You may also like... », dans laquelle sont effectuées des recommandations, qui propose une liste de produits similaires.

Mais actuellement ces produits ne sont pas choisis de manière automatique, Phase Eight fournit une liste fixe de produits à recommander pour chaque produit.

L'objectif de mon stage était de mettre en place une méthode de calcul automatique des produits similaires.

L'idée était de se baser sur les caractéristiques accessibles du produit telles que le nom, la description, le prix ou encore la couleur. Pour cela j'avais à ma disposition un serveur de tests sur lequel se trouvait notamment une copie d'une base de données Infobright contenant les caractéristiques des produits Phase Eight, qu'ont mise en place mes collègues. Elle contenait 139816 produits.

J'ai développé mes programmes en Python, un langage que je n'avais jamais utilisé auparavant, et ai donc découvert au cours de ce stage.

Différentes méthodes

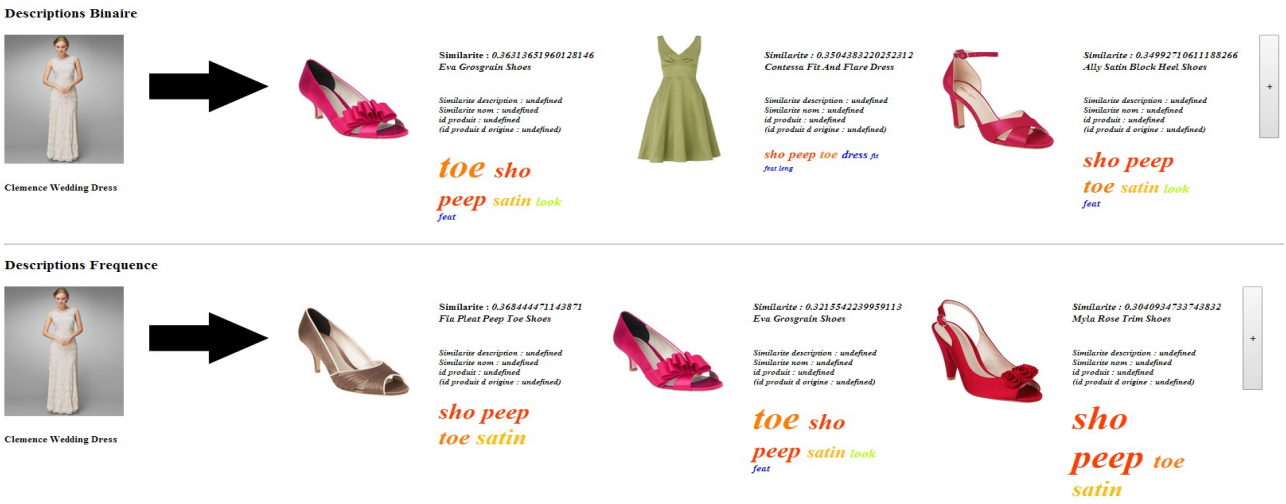
Durant mon travail j'ai mis en place plusieurs versions du calcul des produits similaires à partir d'un produit, progressivement de plus en plus complexes, afin d'améliorer les résultats obtenus.

Les résultats présentés ci-dessous sont tels qu'ils apparaissent au sein d'une interface qu'on m'a demandé de mettre en place, dans laquelle l'utilisateur entre la référence d'un produit et lance le calcul des produits similaires. Les calculs étant relativement longs je me suis limitée dans mes tests aux 1000 premiers produits apparaissant dans la base de données. Les résultats seraient a priori meilleurs lorsqu'effectués sur la totalité de la base de données car on augmenterait les chances de trouver un produit vraiment similaire, et donc on empêcherait l'apparition d' « erreurs » dues à l'absence de produit vraiment similaire en base de données.

- **méthodes simples :**

Les premières versions du calcul que j'ai mises en place sont basées uniquement sur la description du produit. L'idée est de regarder la présence de mots communs dans les deux descriptions.

Dans ces deux versions « simples » du calcul, que je détaillerai plus loin, on observe déjà des résultats satisfaisants pour beaucoup de produits. Toutefois avec certains produits on observe l'apparition d'erreurs.



Ces erreur sont dues à la présence d'éléments non purement descriptifs dans la description. En effet, dans l'exemple ci-dessus il était recommandé dans la description de porter la robe avec des chaussures à talon en satin, et le résultat est que cela parasite le calcul de similarités. Je ne recommande plus plus des robes mais uniquement des chaussures à talon en satin.

Pour éviter ce genre de problèmes, la solution était d'utiliser la catégorie du produit. Si le produit est une robe de mariée, on ne proposera alors plus des chaussures, mais bien des robes de mariée.

La catégorie du produit ne faisant pas partie des données fournies par Phase Eight, il a donc fallu chercher des moyens détournés pour pouvoir la prendre ne compte. D'où les deux méthodes suivantes.

- **méthodes augmentées :**

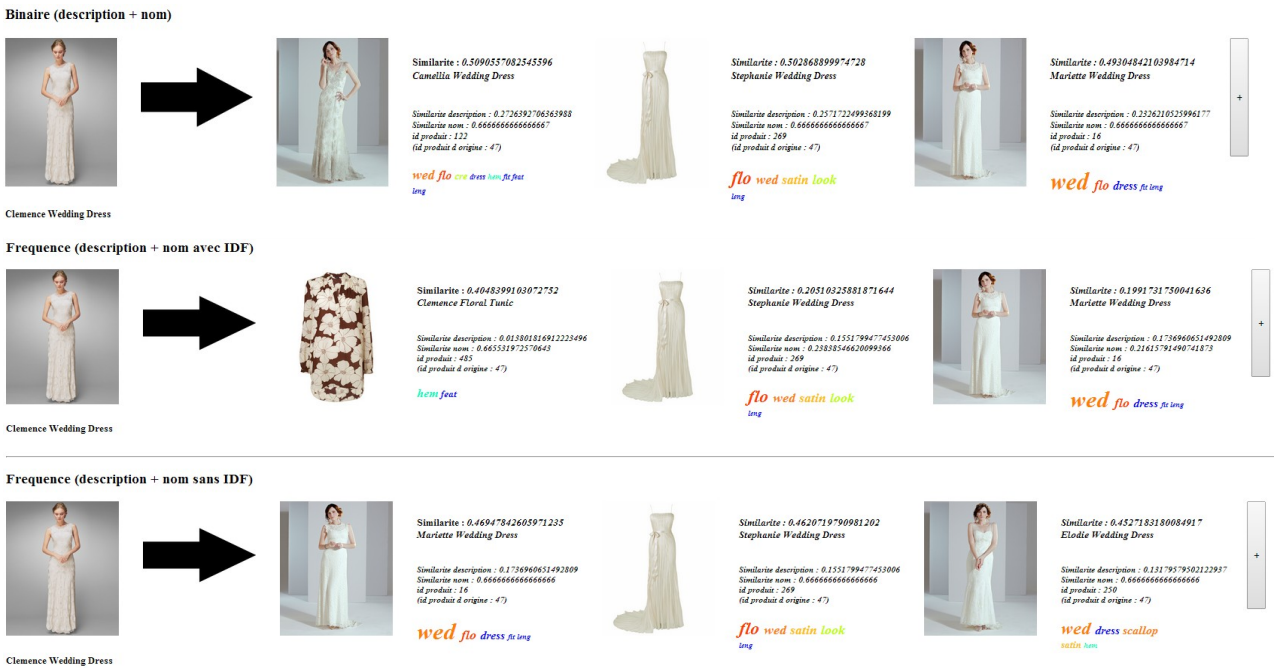
L'idée dans ces méthodes était de prendre en compte les autres informations fournies sur le produit en base de données : le nom, le prix et enfin la couleur. Pour cela je calculais la similarité sur chacune des données, et je faisais une somme pondérée du tout. Pour le prix je me limitais à regarder si les deux produits appartenaient à la même tranche de prix.

Finalement il s'est avéré que les informations sur la couleur étaient peu exploitables, car décrites avec trop de nuances (par exemple pour du vert on trouvait : « vert », « forêt », « lierre », « olive », « émeraude », « jade », « feuille »...). Il devenait presque impossible que deux produits soient considérés comme ayant la même couleur.

Pour le prix il s'est avéré que l'utilisation de tranches parasitait la similarité. J'ai tenté quelques méthodes utilisant à la place une distance entre prix, mais cela s'est avéré peu satisfaisant.

Au final j'ai donc gardé juste le nom et la description.

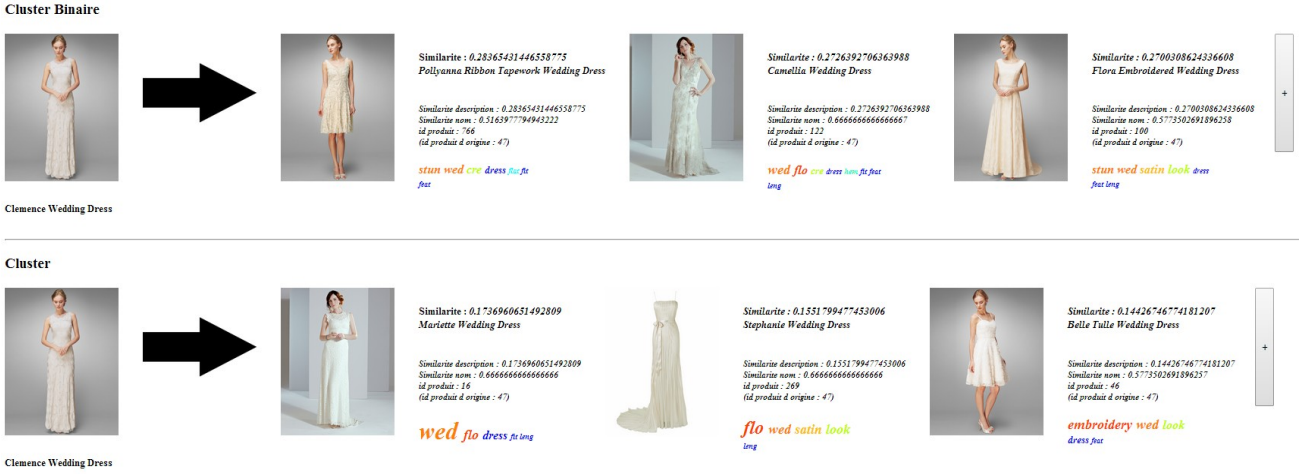
L'ajout du nom dans le calcul de similarité donne des informations sur la catégorie, et règle les problèmes évoqués précédemment.



Ces différentes versions seront expliquées plus loin dans le rapport.

- **méthodes basées sur les clusters :**

Dans cette méthode on n'utilise plus les noms directement comme précédemment. Cette fois on calcule préalablement des clusters à partir des noms des produits, et ensuite on sélectionne lors du calcul des similarités uniquement les produits appartenant aux mêmes clusters préalablement calculés.



Cette méthode présente l'avantage d'éviter que certaines caractéristiques des noms, comme leur longueur, vienne parasiter la similarité descriptive pour plusieurs produits de la même catégorie. Malheureusement les clusters que nous avons pu calculés n'étaient pas parfaits, et engendraient donc d'autres erreurs au niveau de la similarité, que nous montrerons plus loin dans le rapport.

I. Les méthodes simples

1. Traitement des descriptions

La première étape pour calculer les similarités consiste à découper les descriptions en listes de mots (c'est la tokenisation) puis à prendre les racines des mots, les stems (il s'agit donc du stemming). Pour cela j'ai utilisé une bibliothèque en Python qui permet de traiter des langages : NLTK.

Cette bibliothèque met à notre disposition plusieurs algorithmes qui réalisent la tokenisation et le stemming.

a. Tokenisation

Prenons une description :

```
This full length evening dress in a floral textured fabric fits to the figure be
autifully with a feminine fishtail hem. In a striking scarlet shade, this dress
features a soft sweetheart neckline and a V-back. Fastens with a concealed centr
e back zip.
```

Dans la liste des tokenizers disponibles j'en ai retenus deux qui semblaient convenir pour découper une phrase en mots : SpaceTokenizer, qui découpe la phrase selon les espaces, et TreeBank, censé séparer aussi la ponctuation. J'ai fait plusieurs tests dans le but de comparer les durées d'execution et les résultats obtenus.

```
Tokenization SpaceTokenizer
0.00400018692017
['This', 'full', 'length', 'evening', 'dress', 'in', 'a', 'floral', 'textured',
'fabric', 'fits', 'to', 'the', 'figure', 'beautifully', 'with', 'a', 'feminine',
 'fishtail', 'hem.', 'In', 'a', 'striking', 'scarlet', 'shade,', 'this', 'dress'
, 'features', 'a', 'soft', 'sweetheart', 'neckline', 'and', 'a', 'V-back.', 'Fas
tens', 'with', 'a', 'concealed', 'centre', 'back', 'zip.']

Tokenization Treebank
0.00699996948242
['This', 'full', 'length', 'evening', 'dress', 'in', 'a', 'floral', 'textured',
'fabric', 'fits', 'to', 'the', 'figure', 'beautifully', 'with', 'a', 'feminine',
 'fishtail', 'hem.', 'In', 'a', 'striking', 'scarlet', 'shade', ',', 'this', 'dr
ess', 'features', 'a', 'soft', 'sweetheart', 'neckline', 'and', 'a', 'V-back.',
'Fastens', 'with', 'a', 'concealed', 'centre', 'back', 'zip', '.']
```

Au final il s'est avéré que Treebank était un peu plus lent en général, et ne parvenait pas à retirer la ponctuation partout. Par exemple dans l'exemple ci-dessus il reste un point sur « hem. ».

J'ai donc préféré garder SpaceTokenizer et faire un « strip » pour retirer en post-traitement la ponctuation sur les mots de la liste.

b. Stemming

Pour le stemming j'ai testé différents algorithmes fournis par la bibliothèque NLTK. J'en ai finalement retenus deux pertinents : Lancaster et Porter.
Ces deux algorithmes sont adaptés pour le langage anglais, toutefois NLTK fournit d'autres algorithmes si l'on désire utiliser une langue différente, comme le français.

Reprenons la description précédente sous sa nouvelle forme tokenisée :

```
['This', 'full', 'length', 'evening', 'dress', 'in', 'a', 'floral', 'textured',
'fabric', 'fits', 'to', 'the', 'figure', 'beautifully', 'with', 'a', 'feminine',
 'fishtail', 'hem.', 'In', 'a', 'striking', 'scarlet', 'shade,', 'this', 'dress'
, 'features', 'a', 'soft', 'sweetheart', 'neckline', 'and', 'a', 'V-back.', 'Fas
tens', 'with', 'a', 'concealed', 'centre', 'back', 'zip.']
```

Les résultats des tests, effectués avant suppression de la ponctuation, sont les suivants :

```
Stemming Porter
0.00300002098083
['Thi', 'full', 'length', 'even', 'dress', 'in', 'a', 'floral', 'textur', 'fabri
c', 'fit', 'to', 'the', 'figur', 'beauti', 'with', 'a', 'feminin', 'fishtail', '
hem.', 'In', 'a', 'strike', 'scarlet', 'shade,', 'thi', 'dress', 'featur', 'a',
'soft', 'sweetheart', 'necklin', 'and', 'a', 'V-back.', 'Fasten', 'with', 'a', '
conceal', 'centr', 'back', 'zip.']

Stemming Lancaster
0.00899982452393
['thi', 'ful', 'leng', 'ev', 'dress', 'in', 'a', 'flor', 'text', 'fabr', 'fit',
'to', 'the', 'fig', 'beauty', 'with', 'a', 'feminin', 'fishtail', 'hem.', 'in',
'a', 'striking', 'scarlet', 'shade,', 'thi', 'dress', 'feat', 'a', 'soft', 'swee
theart', 'necklin', 'and', 'a', 'v-back.', 'fast', 'with', 'a', 'cont', 'cent',
'back', 'zip.']
```

L'algorithme de Porter est plus rapide, mais semble moins précis, par exemple pour le terme « beautifully », Porter trouve « beauti », tandis que Lancaster identifie que la racine est « beauty ».

J'ai donc finalement utilisé Lancaster. Voici le résultat dans le cas où on a bien retiré la ponctuation avant d'effectuer le stemming :

```
['thi', 'ful', 'leng', 'ev', 'dress', 'in', 'a', 'flor', 'text', 'fabr', 'fit',
'to', 'the', 'fig', 'beauty', 'with', 'a', 'feminin', 'fishtail', 'hem', 'in', '
a', 'striking', 'scarlet', 'shad', 'thi', 'dress', 'feat', 'a', 'soft', 'sweethe
art', 'necklin', 'and', 'a', 'v-back', 'fast', 'with', 'a', 'cont', 'cent', 'bac
k', 'zip']
```

c. Suppression mots parasites

Enfin, la dernière étape consiste à retirer les mots peu importants qui reviennent souvent dans les phrases. Pour cela j'ai utilisé « stopwords », fourni par nltk, qui contient une liste de mots peu significatifs :

```
Stopwords
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yo
urs', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her', 'he
rs', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'thems
elves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am',
'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having
', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'be
cause', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'again
st', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below'
, 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again'
, 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'al
l', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no',
 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'ca
n', 'will', 'just', 'don', 'should', 'now']
```

J'ai donc stemmisée cette liste de mots, puis j'ai retiré de mes descriptions tous les stems appartenant à cette nouvelle liste.

2. Vecteurs

L'étape suivante consiste à stocker chacune des descriptions traitées sous forme de vecteurs en base de données.

Pour cela j'ai créé des bases de données MySQL de taille nombreDeProduits * nombreDeStems.

a. vecteurs binaires :

idProduct	_gown	_gingham	_rol	_edg	_chain	_funct	_tapework	_cobalt	_sleev	_ellips	_skim	_dainty	_go	_cuffs_more	_ros	_encrust	_rom	_graph
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
8	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Chaque ligne est un vecteur qui correspond à une description, chaque colonne correspond à un stem.

Dans le cas des vecteurs binaires on mettra un 1 si le stem est présent et zéro sinon.

b. vecteurs fréquence :

Dans le cas des vecteurs fréquence, si le stem est présent, alors la valeur enregistrée sera TF * IDF.

idProduct	_gown	_gingham	_rol	_edg	_chain	_funct	_tapework	_cobalt	_sleev	_ellips	_skim	_dainty	_go	_cuffs_more	_ros	_encrust	_rom	_graph
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0.0517179	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0.126603	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0.0646474	0	0	0	0	0	0	0	0	0
8	0	0	0	0.57789	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- TF = nombreOccurrencesStem / nombreStems

Par exemple voici les valeurs de TF pour les stems de la description suivante :

```
id :
55

Stems retenus :
['print', 'jersey', 'top', 'cent', 'wrap', 'front', '3/4', 'leng', 'sleev', 'top
', 'fit', 'flat', 'ruch', 'sid']

liste des stems et leur fréquence TF :

print                   wrap                    sleev                   ruch
0.0714285714286         0.0714285714286         0.0714285714286         0.0714285714286

jersey                  front                   top                     sid
0.0714285714286         0.0714285714286         0.142857142857          0.0714285714286

top                     3/4                     fit
0.142857142857          0.0714285714286         0.0714285714286

cent                    leng                    flat
0.0714285714286         0.0714285714286         0.0714285714286
```

on voit que le mot « top » est en double, il a donc un TF deux fois plus élevé que les autres.

- IDF = log(nombreDescriptions / nombreDescriptionsContenantStem)

L'IDF indique l'importance d'un stem par rapport aux autres stems. Par exemple :

```
_sleev                                  _dress                          _edg

nbeDescriptions :                       nbeDescriptions :               nbeDescriptions :
10000.0                                 10000.0                         10000.0

NbeDescriptionsAvecStemPresent :        NbeDescriptionsAvecStemPresent :    NbeDescriptionsAvecStemPresent :
5962                                    3814                            312

IDF :                                   IDF :                           IDF :
0.517179097738                          0.963906585816                  3.46733718417
```

```
_gown                                   _go

nbeDescriptions :                       nbeDescriptions :
10000.0                                 10000.0

NbeDescriptionsAvecStemPresent :        NbeDescriptionsAvecStemPresent :
169                                     1

IDF :                                   IDF :
4.08044165705                           9.21034037198
```

Les termes « sleev » (manche), et « dress » (robe) sont présents dans énormément de descriptions (plus de la moitié pour « sleev »), ce sont donc des mots peu significatifs, qui ne nous aideront pas vraiment à déterminer si deux produits sont similaires. En revanche des termes rares, comme « gown » qui designe des robes plus formelles ou « edge » (bord) se verront accorder plus d'importance, en effet s'ils sont présents dans les deux descriptions c'est beaucoup plus significatif que pour les termes précédents.

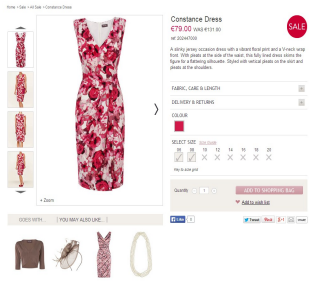
c. Similarité cosinus

Une fois les vecteurs, binaires ou fréquence selon le cas, stockés en base de données, on calcul la similarité en faisant le cosinus de ces deux vecteurs :

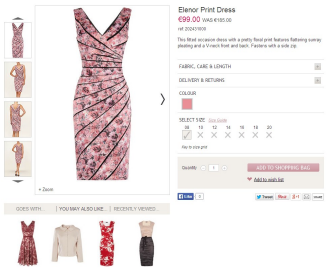
soient v1 et v2 les vecteurs considérés : similarité = v1*v2 / (||v1||*||v2||)

Voici un exemple de comparaison entre deux produits :

```
Id1 :
155
Constance Dress

A slinky jersey occasion dress with a vibrant floral print and a V-neck wrap fro
nt. With pleats at the side of the waist, this fully lined dress skims the figur
e for a flattering silhouette. Styled with vertical pleats on the skirt and plea
ts at the shoulders.

['slinky', 'jersey', 'occas', 'dress', 'vibr', 'flor', 'print', 'v_neck', 'wrap'
, 'front', 'ple', 'sid', 'waist', 'ful', 'lin', 'dress', 'skim', 'fig', 'flat',
'silhouet', 'styl', 'vert', 'ple', 'skirt', 'ple']
```




```
Id2 :
560
Elenor Print Dress

This fitted occasion dress with a pretty floral print features flattering sunray
 pleating and a V-neck front and back. Fastens with a side zip.

['fit', 'occas', 'dress', 'pretty', 'flor', 'print', 'feat', 'flat', 'sunray', '
ple', 'v_neck', 'front', 'back', 'fast', 'sid', 'zip']
```



Voici les stems communs trouvés :

```
Stems communs :
['occas', 'dress', 'flor', 'print',
'flat', 'ple', 'v_neck', 'front', 'sid']
```

Voici les TF – IDF sur les deux descriptions :

```
slinky :     0.04,       3.04913305028
jersey :     0.04,       0.967584026262
occas :     0.04,       2.57046453815
dress :     0.08,       0.963906585816
vibr :     0.04,       4.08637639257
flor :     0.04,       2.06751297081
print :     0.04,       1.92689214322
v_neck :     0.04,       2.28572797593
wrap :     0.04,       2.74887219562
front :     0.04,       1.35751255969
ple :     0.12,       2.79852210427
sid :     0.04,       1.76142426943
waist :     0.04,       1.86175284105
ful :     0.04,       1.06798609513
lin :     0.04,       1.55589714551
dress :     0.08,       0.963906585816
skim :     0.04,       4.05128507276
fig :     0.04,       2.97006452681
flat :     0.04,       1.17506146083
silhouet :     0.04,       1.70374859191
styl :     0.04,       1.79155948923
vert :     0.04,       5.8781358618
ple :     0.12,       2.79852210427
skirt :     0.04,       2.59829933714
ple :     0.12,       2.79852210427
```

```
fit :     0.0625,       0.807436326962
occas :     0.0625,       2.57046453815
dress :     0.0625,       0.963906585816
pretty :     0.0625,       1.89512198222
flor :     0.0625,       2.06751297081
print :     0.0625,       1.92689214322
feat :     0.0625,       0.75950068111
flat :     0.0625,       1.17506146083
sunray :     0.0625,       5.57275421225
ple :     0.0625,       2.79852210427
v_neck :     0.0625,       2.28572797593
front :     0.0625,       1.35751255969
back :     0.0625,       1.37594806907
fast :     0.0625,       1.62353683681
sid :     0.0625,       1.76142426943
zip :     0.0625,       1.59899265457
```

On observe une forte similarité entre les deux produits, voici le résultat dans le cas binaire (à gauche) et fréquence(à droite) :

```
Norme1 :
4.69041575982
Norme2 :
4.0

Similarite :
0.4797016118
```

```
Norme1 :
0.585851324303
Norme2 :
0.55151857852

Similarite :
0.398466898142
```

3. Calcul des similarités

La dernière étape du calcul des similarités, consiste à appliquer la similarité cosinus 2 à 2 pour notre produit étudié et chaque autre produit de la base de données (je me limite ici à 1000 produits). Je garde les N produits les plus similaires (pour l'interface j'ai fixé N à 5), en supprimant les doublons (très nombreux dans la base de données qui m'a été fournie).

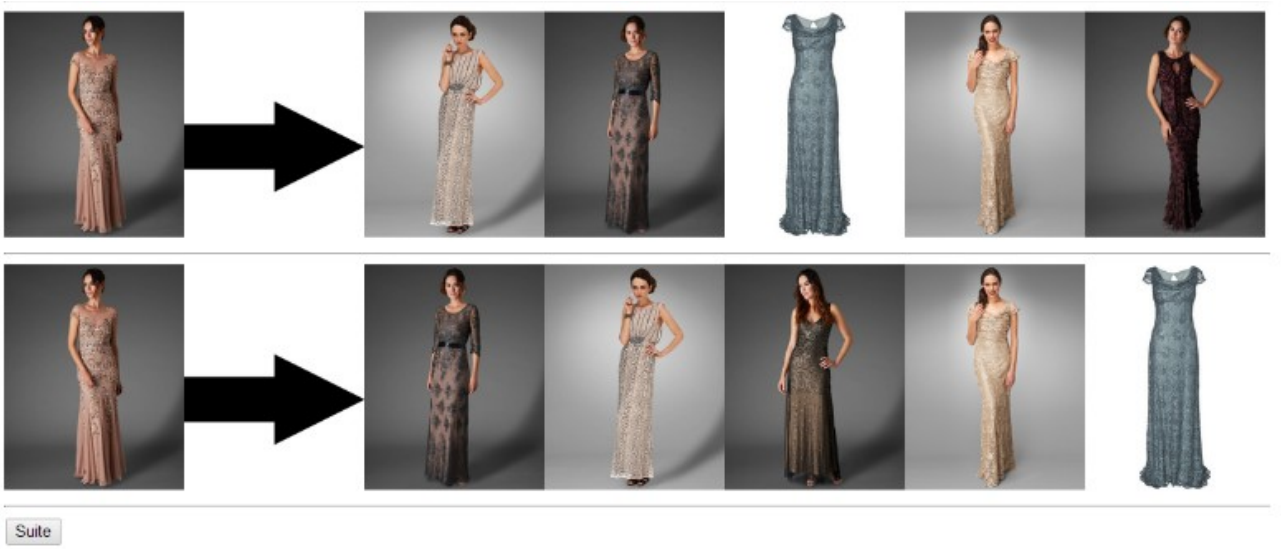
4. Visualisation : l'interface

Une démarche importante chez Devatics est de pouvoir présenter concrètement son travail aux clients, on m'a donc demandé de réaliser une interface pour permette de consulter les similarités des produits de manière interactive.

J'ai donc mis en place un serveur Python ainsi qu'une page html avec Javascript et CSS associés.

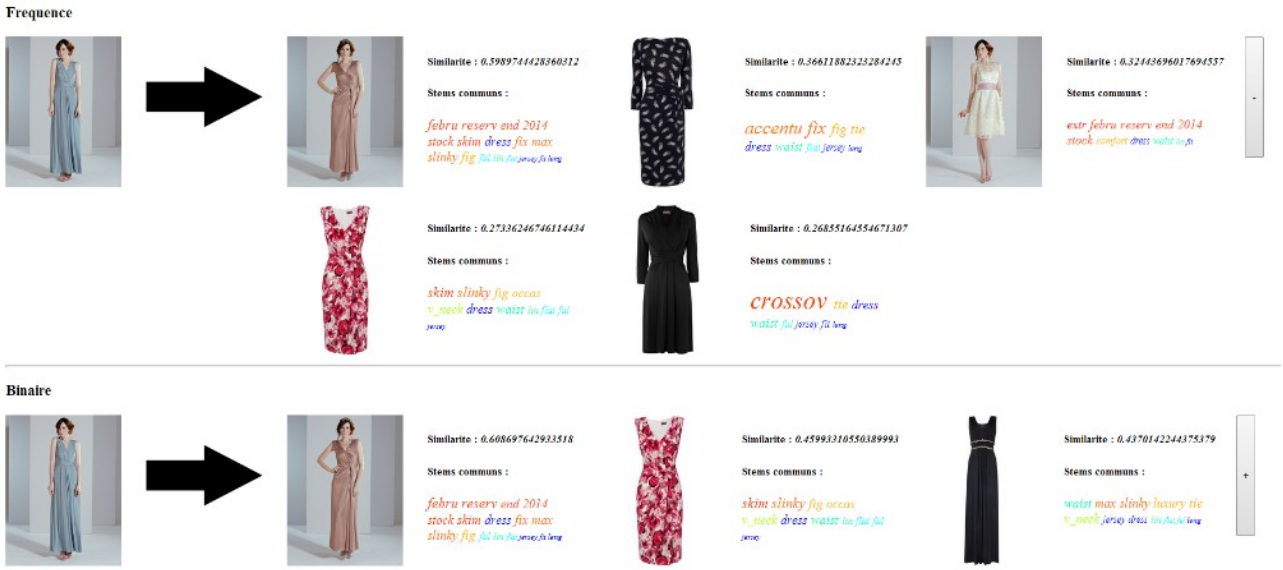
La première version de l'interface tirait un produit au hasard et affichait

simplement les résultats :



On m'a demandé ensuite de permettre au client de choisir la référence du produit étudié, ce que j'ai fait à l'aide d'un formulaire.

J'ai aussi du ajouter à l'affichage la similarité du produit et cacher les produits qui dépassaient de la ligne en les faisant apparaître lorsqu'on clique sur un bouton.



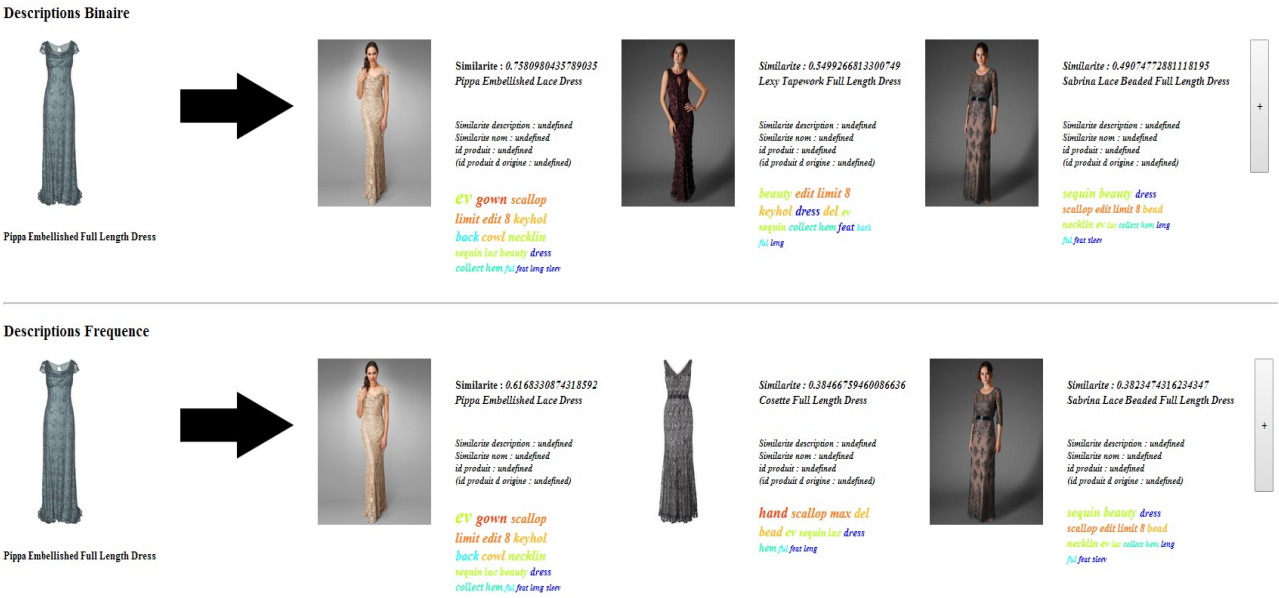
Enfin, on m'a demandé d'ajouter un nuage de mots à partir des stems communs, qui permette d'avoir une compréhension visuelle du TF-IDF de ces différents stems.

Pour cela j'ai choisi d'appliquer une échelle de couleur sur les stems en fonction de leur IDF : de rouge pour un fort IDF à bleu pour un IDF très faible.

J'ai ensuite appliquée une échelle de tailles sur les stems en fonction de leur TF*IDF.

*Sur l'exemple ci-dessus le stem « tul » en orange est en très gros et a donc un fort TF*IDF, et un fort TF puisqu'il est plus gros que « model », en rouge donc avec un très fort IDF (très rare)*

J'ai aussi mise en place une version affichant les diverses similarités de description, nom, prix et couleur pour la version augmentée.



I. Les méthodes augmentées

Comme expliqué dans l'introduction, dans cette méthode, je me suis limitée à rajouter la dimension du nom à celle de la description, les résultats obtenus avec les couleurs et le prix n'étant pas pertinents.

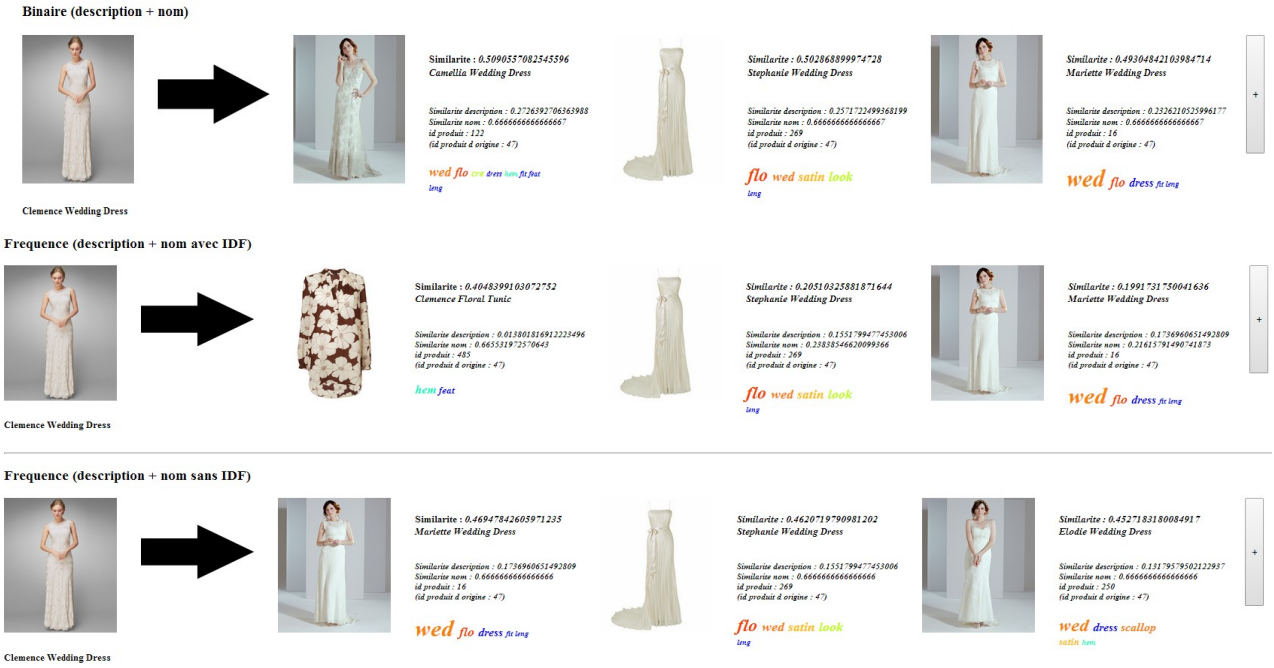
Je calcule donc comme précédemment la similarité sur les descriptions, puis j'utilise la même méthode sur les noms.

Ensuite je fais simplement une somme pondérée de ces deux similarités, avec

un coefficient de 0,6 pour le nom et 0,4 pour la description.

J'ai du retirer l'IDF pour les noms en Frequence, car dans cette méthode on cherche en fait via le nom à trouver une catégorie. Les mots désignant des catégories seront ceux à faible IDF, tandis que les mots rares, comme les noms propres, avec un fort IDF, sont peu importants.

Ce problème avec l'IDF est illustré ci-dessous :
une robe de mariée appelée « Clémence Wedding Dress » se trouvait associée à une tunique « Clémence Floral Tunic ». Ce problème a bien disparu dans le troisième cas, avec les IDF non pris en compte pour les noms.



Toutefois l'influence des noms propres reste quand même présent, et si en plus la description est similaire alors même sans IDF on peut obtenir ceci :

C'est une des raisons pour lesquelles on a désiré mettre en place un vrai calcul des catégories.

L'autre raison est liée à la visualisation, comme nous allons le voir dans le prochain paragraphe.

II. Les méthodes avec clusters

1. Visualisation : les graphes

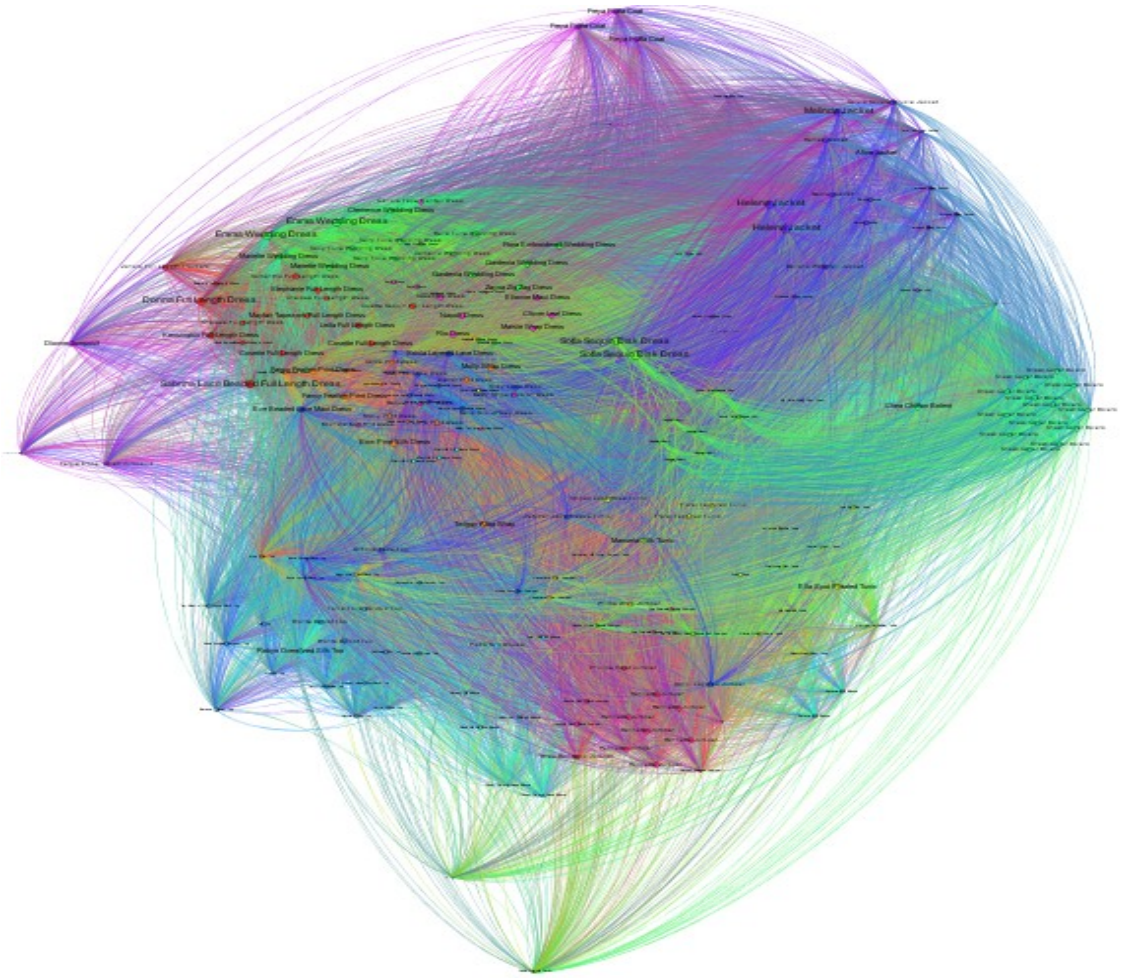
Une autre problématique qui m'a été posée pendant mon stage était de permettre au client de visualiser les résultats des calculs de similarités par des graphes.

Pour cela j'ai utilisé le logiciel Gephi. Ce logiciel permet à partir d'une liste des nœuds et arêtes ainsi que leurs caractéristiques de dessiner le graphe correspondant.

Je créais donc une table en base de données contenant les nœuds de mon graphe (ici les produits), ainsi qu'une autre table avec les arêtes entre deux nœuds et leur poids(correspondant à la similarité des deux produits/noeuds).

A partir de là il fallait régler les paramètres avec Gephi pour tracer le graphe, notamment choisir une spatialisation. J'en ai testé plusieurs et ai fini par en choisir une appelée Force Atlas.

Malheureusement utilisées telles quelles mes données me donnent un résultat in-interprétable :



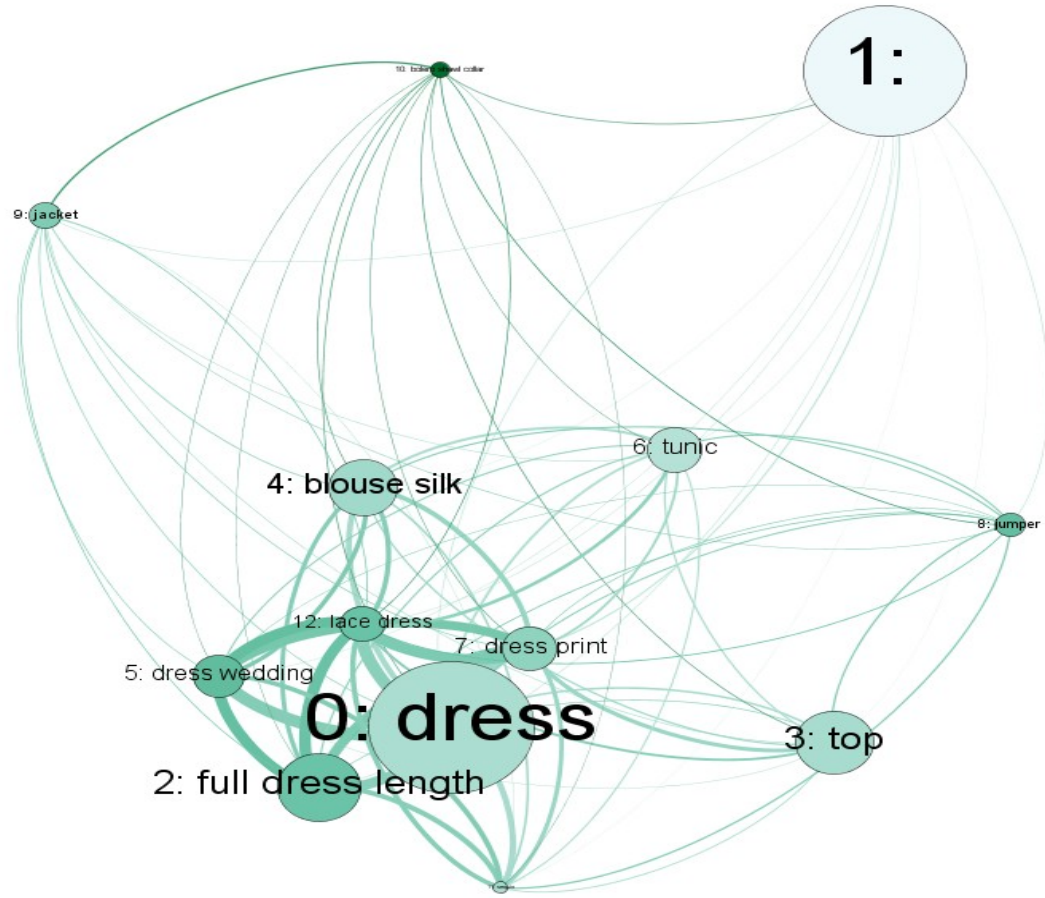
Les couleurs dans le graphe correspondent aux clusters que j'ai calculés avec l'ACP Meanshift comme je l'expliquerai plus loin.

J'ai donc du regrouper les produits par clusters pour permettre d'obtenir un affichage plus lisible.

Chaque nœud est une catégorie, sa taille dépendant du nombre de produits dans la catégorie, et sa couleur correspondant à la similarité intra catégorie. Plus les produits d'une même catégorie sont similaires, plus le nœud sera foncé.

Le label du nœud est calculé de la façon suivante :
Je sélectionne les mots des noms de produits de la catégorie dont l'IDF est supérieur à un certain seuil. (ici 1)

Chaque lien a pour poids la similarité entre les deux catégories qu'il relie.



Pour calculer les similarités intra-catégories et inter-catégories je me suis limitée à 40 produits des par catégorie, en raison des temps de calculs assez élevés.

2. Les clusters

Je vais maintenant expliquer la méthode que j'ai mise en place pour le calcul de catégories.

Ce calcul est effectué à partir des vecteurs fréquence des noms stockés préalablement en base de données.

a. ACP (Analyse en Composantes Principales)

Cette méthode est utile pour réduire les dimensions que l'on utilise.

J'ai au départ une matrice de taille nombreDeProduits * nombreDeStems, et je vais obtenir une matrice de taille nombreDeProduits * q, avec q le nombre de composantes principales désirées.

Voici l'algorithme, que j'ai codé d'abord en Matlab, puis en Python :

*Soit X la matrice de taille nbProduits * nbMots qui contient les tf*idf de chaque mot pour chaque nom de produit.*

Soit Xc la matrice centrée des données (on retire pour chaque produit (ie chaque ligne) la valeur du produit moyen).

*Soit Sigma la matrice de variance covariance Xc'*Xc /nbProduits*

D le vecteur des valeurs propres de Sigma en Python (en Matlab matrice diagonale)

V la matrice des vecteurs propres de Sigma

On trie D pour avoir les q premiers vecteurs propres.

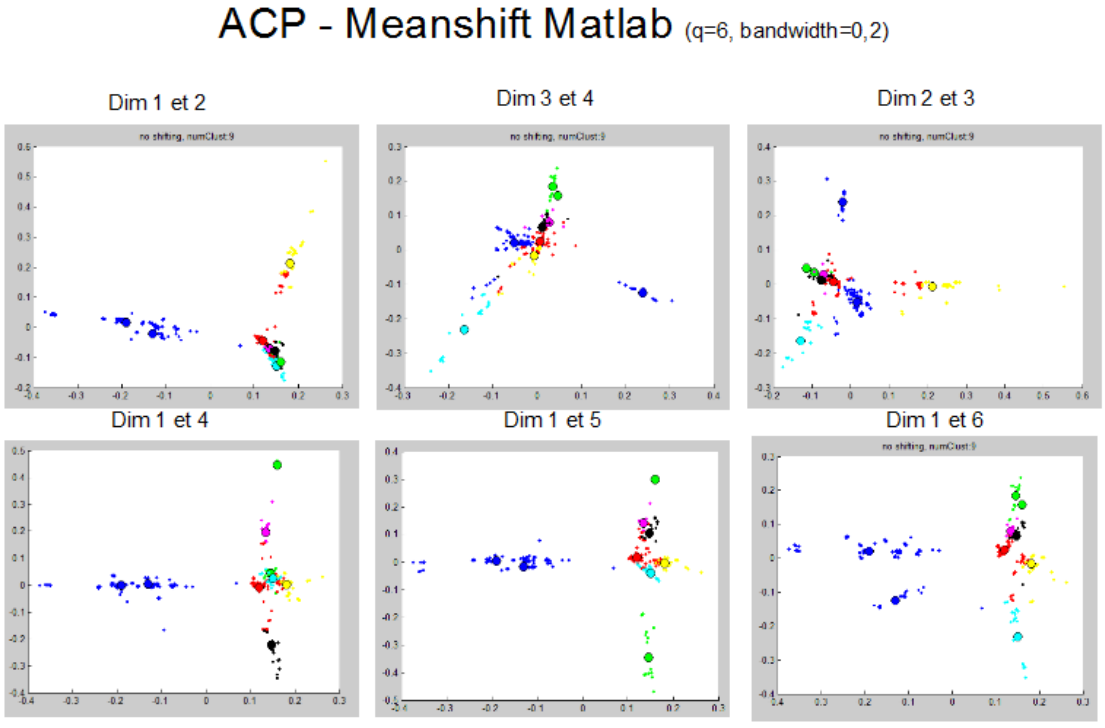
*Soit W la matrice des q premiers vecteurs propres, normalisés, de taille nbStems*q*

*Soit CP la matrice des q composantes principales, CP = Xc*W*

b. Meanshift

J'ai d'abord utilisé une implémentation en Matlab de Meanshift, avant de finalement utiliser une version Python intégrable au reste de mon programme, tirée de la bibliothèque sklearn.

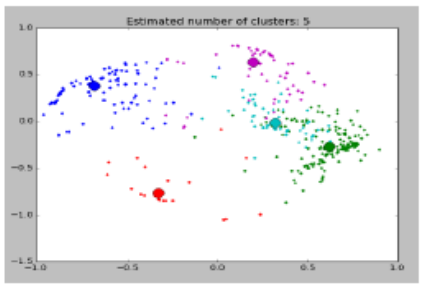
Dans le programme Meanshift je peux faire varier un paramètre, le « bandwidth », ce qui me donne plus ou moins de catégories. Je fais aussi varier le nombre de composantes principales utilisées, « q ».



ACP-Meanshift version normalisée Python

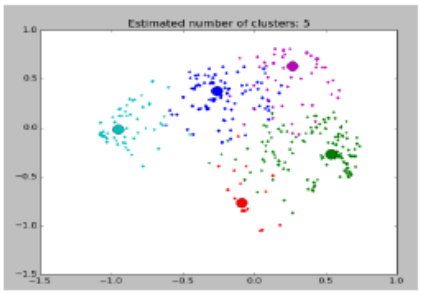
•Q=3, bandwidth=0,5

Dim 1 et 3



Dim 1 et 2

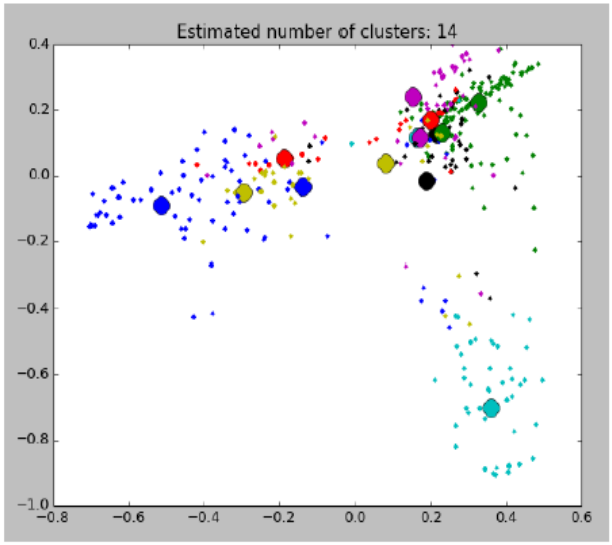
Dim 2 et 3



voici un exemple de catégories trouvées :

ACP-Meanshift (q=10, bandwidth=0,6)

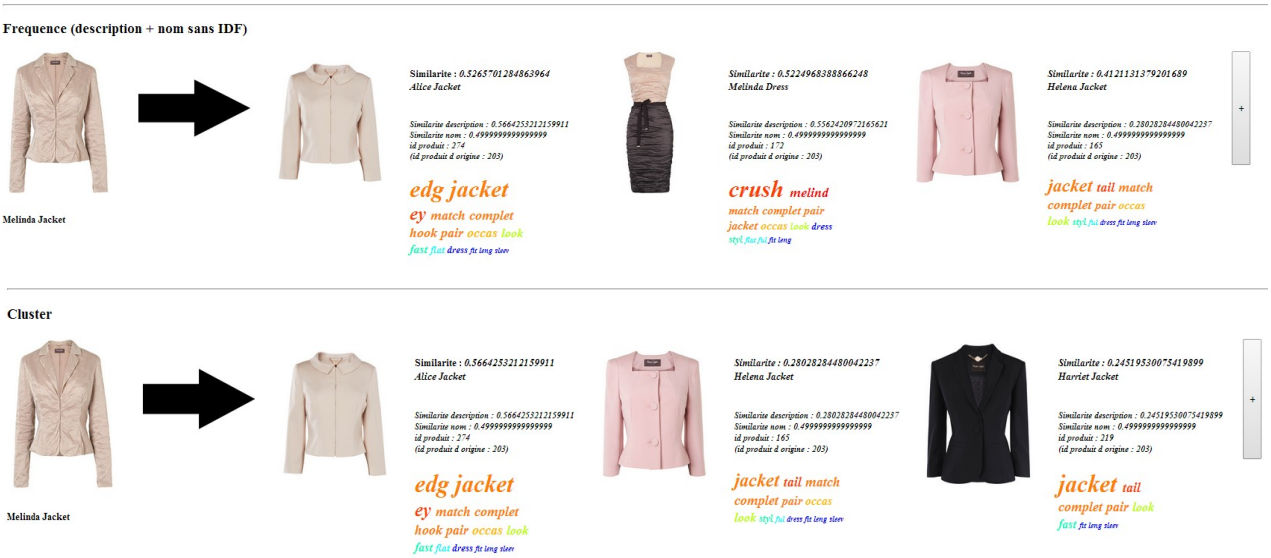
Catégories calculées.
0 : Dress
1 : erreur, Necklace, Coat...
2 : Full Length
3 : Top
4 : Blouse, silk
5 : Wedding Dress
6 : Tunic
7 : Print, Wrap
8 : Jumper
9 : Jacket
10 : Bolero
11 : Sequin
12 :Lace
13 : ? (4 éléments)



c. Similarités par clusters

Les exemples suivants comparent l'algorithme avec nom + description et l'algorithme utilisant directement les clusters.

On observe des améliorations, comme ici où une robe portant le même nom que la veste a bien disparu des propositions.



En revanche, de nouveaux problèmes apparaissent en raison du manque de fiabilité des catégories calculées :

Dans l'exemple ci-dessus, des pantalons longs (« Full length Trousers ») ont été classés dans la même catégorie que les robes longues (« Full length Dress »). Les pantalons qui étaient proposés avant n'apparaissent plus car ils ont été classés dans une autre catégorie, et comme dans l'échantillon qu'on considère il y a peu de pantalons longs, on sera ensuite obligés de proposer des robes longues.

De même ici, le produit « Twiggy Alba Wrap » a été classé dans une catégorie à part par rapport aux produits similaires, car les autres contiennent le terme « top » et ont donc tous été classés dans la catégorie des hauts.

Au final, cette méthode utilisant directement les clusters serait meilleure que celle de l'augmentation car elle supprime des parasites, mais uniquement à la condition d'avoir de bonnes catégories. Les catégories que j'ai calculées durant ce stage ne sont pas assez bonnes pour faire bien fonctionner notre programme, mais la méthode serait utilisable par exemple si le site nous fournit les catégories.

Conclusion

Avant de réaliser ce stage j'avais des appréhensions car je n'avais jamais été confrontée au monde de l'entreprise auparavant. Cette première expérience fut très enrichissante, aussi bien sur le plan technique qu'humainement. J'ai eu l'occasion de travailler avec des technologies très variées, dont une partie furent une découverte. J'ai appris à utiliser Python et javascript, j'ai eu l'occasion de mettre en pratique mes connaissances du langage SQL, j'ai découvert la visualisation de graphes avec Gephi. J'ai aussi eu l'occasion de profiter de la partie relationnelle de l'entreprise, notamment lors d'une ballade en péniche organisée pour rencontrer les clients.

Pour conclure, ce stage me laissera de bons souvenirs, et m'a donné de quoi envisager plus clairement mon avenir professionnel.