UNIVERSITE PARIS DAUPHINE – PSL

LAMSADE

UMR CNRS 7243

CERTIFICAT DATA SCIENCE

Promotion 2021

Mémoire pour l'obtention du certificat en Data Science

Réalisé par :

M. Ballo Blizand SEHI-BI

Docteur en Science économique

17 DECEMBRE 2021

Table des matières

Introduction

Le présent rapport fait suite à la formation reçue dans le cadre du Certificat Data Science. Il a pour but de répondre à un certain de nombre de questions qui nous ont été posées en vue d'évaluer nos connaissances dans le domaine de la data science, et s'articule pour cela autours de deux tâches principales.

La première tâche consiste à faire de la prévision de sentiment à partir d'avis de clients tirés d'une base de produits Amazon. Elle fait donc appel aux technique de ***machine learning*** dont la finalité est de construire un modèle de prédiction après traitement et analyse du texte contenant les avis de l'ensemble des clients.

La seconde tâche vise quant à elle à appliquer les méthodes du ***Big Data*** pour faire du WordCount sous les environnements Linux et Data bricks. Elle est aussi réalisée à partir de la base de données des produits Amazon plus haut cité et contenant trois variables : le nom du produit (name), l'avis du client (review) et son sentiment du produit (sentiment).

Première partie : Tâche 1

Comme indiqué dans l'introduction, cette section consacre l'utilisation des méthodes du *machine learning* en vue de faire de la prévision. Pour ce faire nous tiendrons un notebook basé sur le langage de programmation ***python***.

1. La construction de l'ensemble des caractéristiques

La construction d'une matrice de l'ensemble des caractéristiques a nécessité que nous procédions au préalable à un ensemble d'actions successifs. Celles-ci ont consisté au chargement de la base de données, au nettoyage des observations manquantes, à l'élimination des ponctuations, accents et tout autre caractère inutile, à la racinisation (le *stemming*) et à la tokenisation du texte et enfin à la vectorisation du texte.

1.1 Chargement des données

Le chargement des données et l'analyse préalable des valeurs manquantes par variables ont été réalisés par les requêtes suivantes :

```
#Chargement de la base de données
Produits = pd.read_csv('pr_10k.csv',
                       sep = ';',
                      header = 0,
                      engine = 'python')
#présentation de la base de données
Produits
#Présentation de l'ensemble des données pour les 2 premiers
sentiments positifs
Produits[Produits['sentiment'] == 'positive'].head(2)
#Nombre d'observations manquantes par variables
Produits.isnull().sum()
#Elimination des observations manquantes par variable
Pdts = Produits[Produits["review"].notnull()]
#Nombre d'observations manquantes par variable
Pdts.isnull().sum()
#Nouvelle dimension de la base de données
print(Pdts.shape)
```

Le résultats du nombre d'observations manquantes est :

Variables	Après chargement	Après élimination de obs. manquantes
name	3	3
review	23	0
sentiment	0	0
Dimensions	(9999, 3)	(9976, 3)

Il s'avère que la base de donnée initiale (Produits) comprenait 9999 observations. Toutefois, l'on note 3 et 23 valeurs manquantes au niveau des variables *name* (nom des produits) et *review* (avis des clients) respectivement. Nous avons donc procédé à l'élimination des observations manquantes de la variables *review*. Ce qui nous conduit à une nouvelle dimensions de nos données qui est de 9976 observations au total (cf. tableau ci-dessus).

1.2 Nettoyage des données

Le nettoyage des données va donc consister à l'application d'un ensemble de méthodes – sur la variable explicative nécessaire à la construction de nos différents modèles de prédiction, c'est-à-dire la variables *review* – que nous tenterons de définir :

D'abord, la racinisation des termes (ou ***stemming***) est un précédé qui consiste dans un corpus à ramener chaque mot à sa stricte racine en supprimant les préfixes et les suffixes. Le texte sera de la sorte aussi réduit à sa stricte racine. La fonction utilisée en vue de la racinisation des mots est PorterStemmer() de nltk.

Ensuite la Tokenisation (ou **tokenizing**) ayant pour but le découpage du texte en un ensemble de mot, en d'autres termes, il s'agit de diviser une chaîne de caractère en *tokens*.

A coté de ses deux méthodes nous mettrons en œuvre des méthodes qui visent l'élimination des caractères spéciaux ainsi que celle des mots couts dans le texte.

L'ensemble des différentes requêtes pour ces procédés sont présentées par la suite :

```
#Elimination des Ponctuations, nombres et caractères spéciaux
Pdts['review'] = Pdts['review'].str.replace("[^a-zA-Z#]", " ")
Pdts.head(5)


#Elimination des mots courts
Pdts['review'] = Pdts['review'].apply(lambda x: ' '.join([w for w in
x.split() if len(w)>3]))
#Observation des premières ligne du dataframe
Pdts.head()


#Tokenizing
tokenized_review = Pdts['review'].apply(lambda x: x.split())
#Observation des premières ligne du dataframe
tokenized_review.head()


#Racinisation ou stemming
from nltk.stem.porter import *
stemmer = PorterStemmer()
```

1.3 Compréhension des mots communs utilisés dans les avis : WordCloud

L'une des manières de voir la distribution des sentiments dans l'ensemble des données est de tracer des nuages de mots.

Un nuage de mots est une visualisation dans laquelle les mots les plus fréquents apparaissent en grande taille et les mots moins fréquents en plus petites tailles.

A travers les représentations de nuages de mots qui suivent, il est possible de se faire une idée des mots les plus fréquents dans l'ensemble des avis et selon la nature des sentiments.

Tous les mots

Mots négatifs

Mots positifs

1.4 Construction de la matrice tfidf et terme-documents

Nous employons les deux instructions ci-dessous en vue de générer les matrices termes-document et tfidf nécessaires à l'entrainement de nos modèles. Pour ce faire nous ferons appel à la librairie ***sklearn***.

```
#Chargement des librairies
from sklearn.feature_extraction.text import TfidfVectorizer, CountVect
orizer
import gensim

#Matrice Termes-documents
bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features=1
000, stop_words='english')
bow = bow_vectorizer.fit_transform(Pdts['review'])
bow.shape


#Matrice tfidf
tfidf_vectorizer = TfidfVectorizer(max_df=0.90, min_df=2, max_features
=1000, stop_words='english')
tfidf = tfidf_vectorizer.fit_transform(Pdts['review'])
tfidf.shape
```

2. Les Méthodes de séparation des données

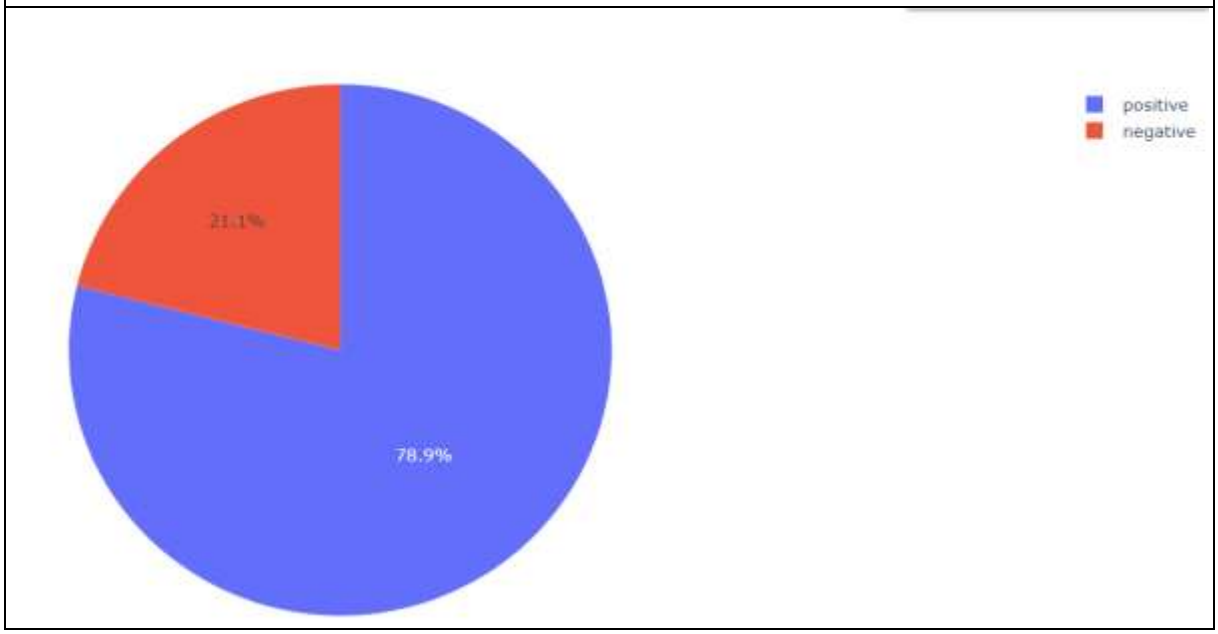
Avant de procéder à l'entrainement de nos données, il est nécessaire de procéder à la séparation des données en sous-échantillons d'entrainement et de test . Nous présenterons trois approches pour le faire : dans les deux premières, les répartitions **naïve** et **stratifiée**, le sous-échantillon d'entrainement constitue (2/3) de la population et celui de test (1/3) . En revanche, dans la dernière approche (boostrap), l'échantillon est subdivisé en 10 sous-échantillons dont à chaque fois (9/10) de l'échantillon est utilisé pour l'entrainement et (1/10) pour le test.

Par ailleurs, la répartition des sentiments dans la variable *sentiment* peut être déterminante dans la stratégie à adopter pour le partitionnement de l'échantillon en sous échantillons d'entrainement et de test. Le tableau ci-dessous nous permet de le voir.

```
#Comptage des sentiments selon le type de sentiment
Pdts["sentiment"].value_counts()
```

Resultats

positive 7869
negative 2107



La répartition **naïve** veut que l'on tire de manière aléatoire 66% des observations de notre échantillon total qui constituerons notre sous-échantillon d'entrainement tandis que le reste (33%) sera utilisé pour tester le modèle que nous aurons choisi d'entrainer. Cela exige cependant que notre échantillon soit bien équilibré. Ce qui n'est pas le cas de notre échantillon.

En effet, la figure ci-dessus montre que nous avons 21% de sentiments négatifs et plus de 78% de sentiment positif. Nous présentons tout de même les résultats de la répartition naïve.

Requête

```
#Sélection des échantillons pour l'apprentissage et le test
from sklearn.model_selection import train_test_split
XTrain, XTest, yTrain, yTest = train_test_split(tfidf, Pdts['sentiment
'], train_size = 0.66, random_state = 42)
```

Ainsi la répartition naïve nous donnes des répartitions quasiment proches de l'échantillon totale. Etant donné que ces résultats sont quelque peu différents, cette approche ne sera pas utilisée pour la suite de l'analyse.

Résultats

Sentiment	Echantillon d'entrainement	Echantillon test
Négatif	0.207928	0.217571
Positif	0.792072	0.782429

Une autre approche qualifié de « **stratifiée »** pourrait être de maintenir des structures pour les sous-échantillons d'entrainement et de test conforment à celle de notre échantillon d'origine. Par l'option ***stratify***, il s'agit d'imposer à l'algorithme générateur des bases d'entrainement et test de respecter les mêmes proportions pour le variable sentiment que l'échantillon d'origine.

Requête

```
#Sélection des échantion pour l'apprentissage et le test
from sklearn.model_selection import train_test_split
XTrain1, XTest1= train_test_split(Pdts, train_size=0.66,random_state=4
2,stratify=Pdts['sentiment'])
```

Contrairement à la méthode naïve, nous trouvons la même répartition que dans les données initiale.

Résultats

Sentiment	Echantillon d'entrainement	Echantillon test
Négatif	0.21127	0.211085
Positif	0.78873	0.788915

Une troisième approche de répartition peut toutefois sembler être la bonne méthode pour l'entrainement des données et leur test. En effet, la performance d'un modèle repose sur l'analyse de métriques qui en évaluent la précision. La précision pour un ensemble de test

pouvant varier lorsque nous l'appliquons sur une autre échantillon test, le modèle peut paraître non fiable. Ainsi la répartition par boostrap utilisant le k-fold Cross validation (CV) peut constituer une solution au problème en divisant la base en sous-échantillons et en s'assurant de l'utilisation de chaque sous-échantillon comme ensemble test à un moment donné.

3. Présentation, entrainement et évaluation des méthodes d'apprentissage

L'objectif de notre travail est de construire un modèle qui nous permette de prédire avec une meilleure précision la classe des avis ne faisant pas partie de l'ensemble de nos données. Pour ce faire, il nous faut entrainer des modèles à partir de notre échantillon d'apprentissage pour ensuite les tester à l'aide de l'échantillon test. Nous avons choisi comme méthodes d'apprentissage celles dites d'apprentissage supervisé que sont : l'**arbre de décision**, les **Support Vector Machines** (SVM) et le **random forest (Foret aléatoire).**

3.1 Arbre de décision[1]

L'apprentissage par arbre de décision est l'une des approches de modélisation prédictive utilisées en statistiques, en exploration des données (***data mining***) et en apprentissage automatisé (***machine learning***). Il utilise un arbre de décision (en tant que modèle prédictif) pour passer des observations sur un élément (représentés dans les branches) aux conclusions sur la valeur cible de l'élément (représenté dans les feuilles). Les modèles arborescents où la variable cible peut prendre un ensemble discret de valeurs sont appelés arbre de classification ; dans ces structures arborescentes, les feuilles représentent les étiquettes de classe et les branches représentent les conjonctions de caractéristiques qui mènent à ces étiquettes de classe. Les arbres de décisions dans lesquels la variable cible peut prendre des valeurs continues (généralement des nombres réels) sont appelés arbres de régression. Les arbres de décisions font partie des algorithmes d'apprentissage automatique les plus populaires en raison de leur intelligibilité et de leur simplicité.

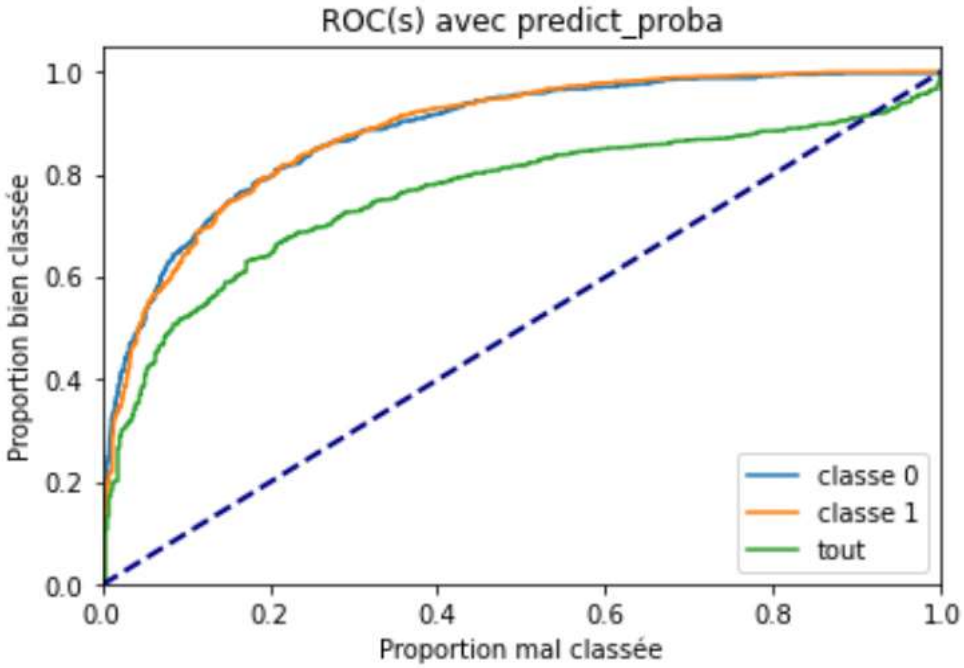
Requête

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
clf_AD=AdaBoostClassifier(DecisionTreeClassifier(min_samples_leaf=5),
 algorithm="SAMME", n_estimators=100)
clf_AD.fit(XTrain,yTrain)
```

Accuracy = 0.861144

[1] https://en.wikipedia.org/wiki/Decision_tree_learning

ROC



3.2 Random forest[2]

L'idée derrière la méthode ***random forest*** est qu'elle fait pousser de nombreux arbres de classification simple. Ainsi pour classer un objet à partir d'un vecteur d'entrée, on place le vecteur d'entrée dans chacun des arbres de la forêt. Chaque arbre donne une classification, et on dit que l'arbre « vote » pour cette classe. Au final, la forêt choisit la classification ayant le plus de votes sur tous les arbres de la forêt.

Requête

```
# Random forest
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
clf_RF=RandomForestClassifier(n_estimators=100)
clf_RF.fit(XTrain,yTrain)
```

Accuracy = 0.858491

[2] https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

3.3 Support Vector Machine (SVM)

Parmi les méthodes d'apprentissage, le SVM est une méthode offrant une précision très élevée par rapport à d'autres classificateurs tels que la régression logistique et même les arbres de décision. Cette méthode est connu pour son astuce de noyau pour gérer les espaces d'entrée non linéaire. Il est utilisé dans une variété d'applications telles que la détection des visages, la détection des intrusions, la classification des gènes etc. A partir de l'algorithme SVM, le classificateur sépare les ponts de données en utilisant un hyperplan avec la plus grande marge. C'est pourquoi un classificateur SVM est également connu comme un classificateur discriminant. Le SVM trouve un hyperplan optimal qui aide à classer les nouveaux points données.

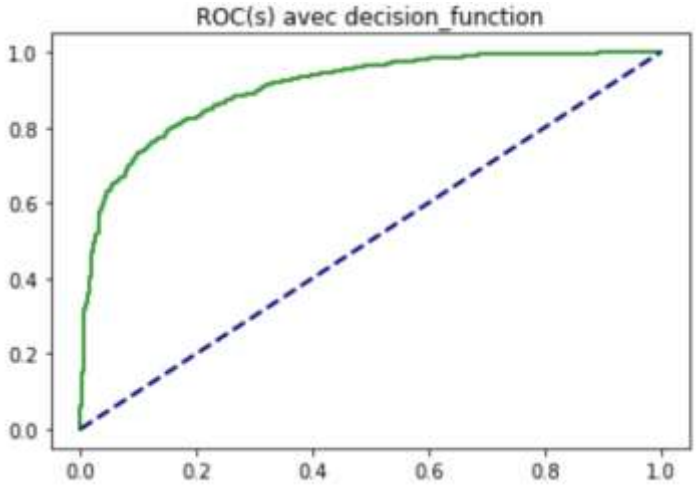
Pour sa mise en œuvre nous utiliserons la librairie scikit-learn comme les méthodes précédentes.

Requête

```
from sklearn.svm import SVC
clf_SVC= SVC(kernel='linear')
clf_SVC.fit(XTrain,yTrain)
```

Accuracy = 0.867040

AUC et ROC



5. Conclusion partielle

A l'issue de notre analyse, il semble que le support vecteur machine SVM soit la plus performante des méthodes appliquées sur un échantillon équilibré. Elle offre en effet une amélioration de la prédiction du sentiment.

Deuxième partie : Tâche 2

Comme indiqué également dans l'introduction, cette partie consacre le Big data . Elle a pour but de ressourdre les problèmes de WordCount dans les environnements linux avec MapReduce et Data bricks . Les questions étant pour la plupart indépendante, elles pourront être résolues séparément.

1. Dépôt du fichier .csv dans ubuntu

Les instructions qui suivent nous ont permettent de télécharger les données dans linux puis de les charger dans pyspark.

Requêtes

```
#Téléchargement de la base
wget https://www.dropbox.com/s/fjtimrh3nbm4fgq/pr_10k.csv?dl=0


#Modification de la base
mv pr_10k.csv?dl=0 prodts.csv

#lancement de pyspark
spark-2.4.0-bin-hadoop2.7/bin/pyspark

#Chargement de la base de données dans pyspark.
>>> prodts = sc.textFile("/home/sehibi/prodts.csv")

>>> prodts.take(3)
```

Resultats

[u'name;review;sentiment', u'Annas Dream Full Quilt with 2 Shams;Very soft and comfortable and warmer than it looks...fit the full size bed perfectly...would recommend to anyone looking for this type of quilt;positive', u"Stop Pacifier Sucking without tears with Thumbuddy To Love's Binky Fairy Puppet and Adorable Book;This is a product well worth the purchase. I have not found anything else like this, and it is a positive, ingenious approach to losing the binky. What I love most about this product is how much ownership my daughter has in getting rid of the binky. She is so proud of herself, and loves her little fairy. I love the artwork, the chart in the back, and the clever approach of this tool.;positive"]

2. Application de WordCount au fichier .csv

Avant de procéder au décompte des mots nous effectuons d'abord un nettoyage de nos données.

2.1 Nettoyage de la base

C'est le but des instructions ci-dessous.

Requêtes

```
#Selection de l'entête
>>> header = prodts.first()

#Affichage de la sélection
>>> print(header)

#suppression de l'entête
>>> prodts1 = prodts.filter(lambda x : x != header)

#Sélection du texte portant sur les avis et séparation des mots du texte
>>> review = prodts1.map(lambda x : x.split(";")).map(lambda x : x[1])


#Suppression des caractères liants les mots
>>> review_well = review.map(lambda x : x.replace('...','   ') or x.replace('-',' ') or x.replace('..','  ')
or x.replace('&',' '))
```

Resultats d'avant la suppression des caractères liants les mots

[u'Very soft and comfortable and warmer than it looks...fit the full size bed perfectly...would recommend to anyone looking for this type of quilt', u'This is a product well worth the purchase. I have not found anything else like this, and it is a positive, ingenious approach to losing the binky. What I love most about this product is how much ownership my daughter has in getting rid of the binky. She is so proud of herself, and loves her little fairy. I love the artwork, the chart in the back, and the clever approach of this tool.']

2.2 WordCount

Les instructions qui suivent permette enfin de faire le WordCount .

Requêtes

```
# Pour effectuer le WordCount
>>> wc = review_well.flatMap(lambda x : x.strip().split()).map(lambda x
(x,1)).reduceByKey(lambda x,y : x+y)
>>> wc.take(50)
```

Resultats

[(u'considered,', 2), (u'considered.', 2), (u'operation.&quot,', 1), (u'Kohls,', 1), (u'hanging', 83), (u'everything.I', 1), (u"weren't).", 1), (u'pre-mature', 1), (u'gad', 1), (u'LAST', 1), (u'persisted.3.', 1), (u'Intially', 2), (u'balmex,', 1), (u'Euro', 8), (u'wood!', 1), (u'wood)', 1), (u'wood,', 6), (u'wood.', 5), (u'Ingenious!', 1), (u'PS2,', 1), (u'Ordered', 6), (u'screaming', 63), (u'wooden', 49), (u'halfway,', 1), (u'SNOW!!!!', 1), (u'halfway.', 2), (u'remoisten', 3), (u"do.I'm", 2), (u'crotch', 4), (u'ocean,', 2), (u'LeapFrog:', 1), (u'seven.', 1), (u'potty,i', 1), (u'HAPPENED', 1), (u'snuggley', 1), (u'job.The', 2), (u'FIGURED', 2), (u'(except', 11), (u'gaskets', 7), (u'snuggles', 1), (u'scrapes', 1), (u'27.', 1), (u'270', 1), (u'bannister', 1), (u'sooth', 9), (u'chin,', 5), (u'scraped', 2), (u'snuggled', 7), (u')It', 1), (u'errors', 1)]

3. Calcul du nombre d'avis selon l'étiquette

Pour la suite de la partie 2, le reste des questions sera traité dans un notebook généré à partir de Data bricks. Les instructions suivantes ont permis de le générer et présenter un dataframe à cet effet.

3.1 Création d'un dataframe à partir de notebook Data brics

```
# File location and type
file_location = "/FileStore/tables/pr_10k-8.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ";"

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

3.2 Calcul des avis

Le nombre d'avis positif par produit :

Requêtes

```
from pyspark.sql.functions import desc
q1 = df.where("sentiment == 'positive'").select("name")
q2 = q1.groupBy("name").count().sort(desc("count"))
q2.show(10)
```

Résultats :

name	count
Leachco Snoogle T...	338
Baby Trend Diaper...	248
BABYBJORN Potty C...	209
The First Year's ...	195
PRIMO EuroBath, P...	162

Munchkin Mozart M...	156
The First Years H...	153
…	…

Le nombre d'avis négatifs par produit :

Requêtes

```
from pyspark.sql.functions import desc
q4 = df.where("sentiment == 'negative'").select("name")
q5 = q4.groupBy("name").count().sort(desc("count"))
q5.show(10)
```

Resultats :

name	count
Safety 1st Deluxe...	91
Philips Avent 3 P...	78
Baby Trend Diaper...	50
Playtex Diaper Ge...	38
Prince Lionheart ...	36
The First Years B...	35
Soothing Dreams M...	34
…	…

Enregistrement avec des avis vides :

Requêtes

```
q6 = df.filter(df.review.isNull()).select("name")
q6.count()
```

Résultats :
23

Conclusion

Notre travail a eu pour but l'analyse des sentiments à partir d'avis sur des produits Amazon. Pour ce faire nous avons mis en œuvre deux types de méthodes : celles du *machine learning* et celles du big Data. En particulier, la première méthode visait la détermination d'un modèle de prédiction capable de prévoir avec une grande précision. Dans ce cas nos résultats ploient en faveur de la technique du Support vecteur Machine (SVM). Les méthodes du Big data nous ont quant à elle amené à mobiliser des environnements tel que spark sur linux et MapReduce dans le but de faire du WordCount.