

# CATÉGORISATION DE PRODUITS

## ATELIERS-BIG-DATA

---

Brendan Guillouet

11 Décembre 2017

Institut National des Sciences Appliquées

Introduction

Nettoyage des données

Vectorisation

Apprentissage

TP

# INTRODUCTION

---

Multiple **domaines de recherche** et une très grande variété d'**applications** :

- Recherche d'information.
  - *Moteur de recherche (Google, Yahoo).*
- Traitement Automatique du langage.
  - *ChatBot.*
- Reconnaissance de pattern.
  - *Scrawling de page.*
- Analyse de sentiments.
  - *Marketing.*
- Désambiguïsation.
  - *Sécurité.*

# LA CATÉGORISATION DE PRODUITS

## OBJECTIF :

Automatiser la catégorisation des produits dans l'arborescence du site. (*Concours DataScience.net*)

<https://www.data-science.net/fr/challenge/20/details>

- 47 Catégories de niveau 1.
- 536 Catégories de niveau 2.
- 5789 Catégories de niveau 3.

## DIFFICULTÉS :

- Gérer des gros volumes de données (**Big Data**) :
  - *dizaines de millions de produits.*
- Données **réelles** :
  - *étape de nettoyage importante.*
- Appréhender un nouveau type de données : **textuelles**.

Données issues du concours proposé par **Cdiscount** et disponible sur **Datascience**.

Fichier d'apprentissage de 15.786.885 produits.

| Champ               | Type de données | Description                   |
|---------------------|-----------------|-------------------------------|
| Identifiant produit | String          | Identifiant unique du produit |
| Catégorie 1         | Int             | Catégorie de niveau 1         |
| Catégorie 2         | Int             | Catégorie de niveau 2         |
| Catégorie 3         | Int             | Catégorie de niveau 3         |
| Description         | String          | Description produit           |
| Libelle             | String          | Description courte            |
| Marque              | String          | Marque du produit             |

**TABLE 1** – Données Cdiscount.

| Catégorie1        | Description                        |
|-------------------|------------------------------------|
| CULTURE - JEUX    | playstation 4 et jeu pes 4         |
| CULTURE - JEUX    | xbox 360 et call of duty           |
| CULTURE - JEUX    | manette pour xbox et playstation 4 |
| JARDIN - PISCINE  | grande pelle verte                 |
| JARDIN - PISCINE  | pompe a eau pour jardin            |
| AUTO - MOTO (NEW) | 360 casque avec visière            |
| AUTO - MOTO (NEW) | pompe a eau et gaz                 |

TABLE 2 – Exemple de catalogue d'apprentissage.

## DIFFICULTÉS :

- **Bruits** liés aux fautes d'orthographe, aux accords, à la conjugaison.
- Nombreux termes **non significatifs**.
- Termes significatifs liés **au contexte**.
- **Transcription** aux outils machine learning.
- Décision liée au **contexte**.
- Traitement **différent** d'une langue à l'autre.

⇒ Étape de pré-traitement des données très importante.



# NETTOYAGE DES DONNÉES

---

**PROBLÈME :** Des termes peuvent-être écrits de **différentes façons** (accents, conjugaison, genre, pluriels..) et néanmoins avoir le **même sens**.

**SOLUTION :** Remplacer les mot par leur **racine**.

**EXEMPLE :**

- épée, épee, epée = epe
- vert, verts, verte, vertes = vert
- mange, manger, mangez, mangent = mang

De nombreux algorithmes différents propre aux langues étudiées  
Algorithme utilisé : *Snowball (Porter)*

## PROBLÈME :

Des termes très usuels, donc non différenciateurs, peuvent perturber l'apprentissage.

## SOLUTION :

Lors de l'étape de preprocessing, les mots les plus communs sont supprimés à partir d'une liste de **Stopword** :

(["a", "afin", "ai", "ainsi", "après", "attendu", "au", "aujourd", "auquel",  
"aussi", "autre", "autres", "aux", "auxquelles", "auxquels", "avait", "avant",  
"avec", "avoir", "c", "car", "ce", "ceci", "cela", "celle", "celles", "celui",  
"cependant", "certain", "certaine", "certaines", "certains", "ces", "cet", "cette",  
"ceux", "chez", "ci",])

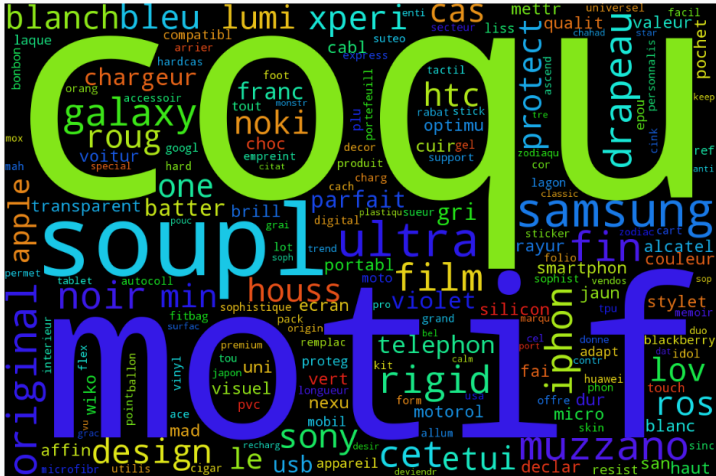
⇒ Également propre à la langue !

- Suppression de code HTML (*BeautifulSoup*).
- Suppression de la ponctuation.
- Incrémenter les stopwords avec de la connaissance métier.
- Suppression des caractéristiques numériques (sauf pour les marques)  
⇒ Importance de la connaissance métier!





## ILLUSTRATION - CATÉGORIE "TÉLÉPHONIE - GPS"



Ajout de stowpords liés au contexte.

- **NLTK** (*Python, Spark*) : Traitement du langage (racinisation, stopwords, ...).
- **Lucène** (*Java*) : Librairie d'indexation et de recherche de texte.
- **BeautifulSoup** : Suppression des balises HTML.
- **Regex** : Langage de recherche de texte.



# VECTORISATION

---

- Transformer la liste de mots sous un format **interprétable** par les différents algorithmes d'apprentissage.
- Gérer le très **grand nombre** de features.
  - Exemple sur 21.543 lignes de la catégorie "TÉLÉPHONIE-GPS"
  - 24.486 mots uniques -> 8.384 après nettoyage.
- Choisir des poids **significatifs**.

**Vectoriser** les descriptions tout en **réduisant** l'espace de stockage.

$X$

$\Rightarrow$

$\phi$

Vecteurs de tailles aléatoires.

Très grand nombre de features.

Taille inconnue

Vecteurs de tailles fixes.

Taille fixé à l'avance : **n\_hash**.

- Éviter les collisions.
- Accélérer la vitesse de calcul.
- Non aléatoire.

## Definition (Fonction de hashage - 1)

$$h: \mathbb{N} \rightarrow \{1, \dots, n\_hash\}$$
$$i \mapsto j = h(i).$$

## Definition (Fonction de hashage - 2)

$$\xi: \mathbb{N} \rightarrow \{1, -1\}$$
$$i \mapsto j = \xi(i).$$

## Definition (Hashed Feature Map)

$$\phi_j^{\xi, h}(x) = \sum_{i \text{ s.t. } h(i)=j} \xi(i)x_i$$

## PROBLÉMATIQUE :

Certains mots n'ont pas le même sens en fonction du contexte.

- *Short de bain*  $\neq$  *Short*  $\neq$  *bain*

## SOLUTION :

On ne considère pas que les mots uniques (*unigram*) mais les couples de mots (*bigram*), ou toutes associations de  $n$  mots ( $n$ -gram).

- Résout les ambiguïté du langage.
- Explosion de la taille des vecteurs
  - Exemple sur 21.543 lignes de la catégorie "TELEPHONIE-GPS"
  - 8.384 *unigram*, 50.012 *bigram*, 90.854 *trigram*...

Assigner un **score d'importance** d'un mot, ou d'une association de mots, dans un document **relativement à un ensemble de document**.

- $t$  : mot ou association de mot.
- $d$  : un document.
- $D$  : un ensemble de document.

## Definition (Formule Générale TF-IDF)

$$tfidf(t, d) = tf(t, d) \times idf(t, D)$$

- $tf(t, d)$  : *Term-Frequency* Nombre d'occurrence du terme  $t$  dans le document  $d$ .
- $idf(t, D)$  : *Inverse-Document-Frequency* Mesure l'importance du terme  $t$  dans l'ensemble des documents  $D$ .

## FORMULE DU TF

Le terme  $tf(t, d)$  est généralement défini comme le nombre d'occurrences du terme  $t$  dans le document  $d$  :

$$tf(t, d) = f_{t,d}$$

Cette définition est celle utilisée dans les librairies *scikit-learn* de *Python* et *Mllib* de *Spark*.

Cependant des variations peuvent exister :

|                                  |   |
|----------------------------------|---|
| binaire                          | 0, 1  |
| normalisation logarithmique      | $1 + \log(f_{t,d})$   |
| normalisation « 0.5 » par le max | $0.5 + 0.5 \times \frac{f_{t,d}}{\max_{t' \in d} f_{t',d}}$   |
| normalisation par le max         | $K + (1 - K) \times \frac{f_{t,d}}{\max_{t' \in d} f_{t',d}}$ |

**TABLE 3** – Variante du TF (source wikipedia)

La formule du terme *IDF* varie également d'une implémentation à l'autre.

|                                     |                              |
|-------------------------------------|------------------------------|
| $\log(\frac{N_D}{DF(t,D)})$         |                              |
| $\log(\frac{N_D+1}{DF(t,D)+1})$     | <i>MLib (Spark)</i>          |
| $\log(\frac{N_D+1}{DF(t,D)+1}) + 1$ | <i>scikit-learn (Python)</i> |

**TABLE 4** – Variante de l'IDF

- $N_D$  : Nombre de documents.
- $DF(t, D)$  : Nombre de documents dans lequel le terme  $t$  apparaît.



- Les fonctions de **Hashage** et de **TF-IDF** sont appliquées sur le jeu de données d'apprentissage.
- La même fonction de **Hashage** est utilisée sur le jeu de données test.
- Les termes de **TF** entre un mot  $t$  et un document  $d$  sont recalculés pour le jeu de données test
- Les termes d'**IDF** calculés lors de l'apprentissage sont réutilisés.

# APPRENTISSAGE

---

- Par défaut, *scikit-learn*, *MMLib* utilisent des algorithmes "one-vs-all"
- Dans *scikit-learn* différents solveurs sont disponibles (*liblinear*, *LBFGS*, *sag*...).

- Les 4 meilleures solutions ont utilisées des méthodes linéaires Goutorbe et al. [2016]
- Solutions à plusieurs niveaux.
  - *On classifie parmi les 1ère catégories..*
  - *...puis parmi les catégories de niveau 4*
- Implémentation de la solution N°2 disponible :  
<https://github.com/ngaude/cdiscount>

TP



#### 4 Notebooks :

- *Atelier-Cdiscount-python-2.ipynb* (py27)
- *Atelier-Cdiscount-python-3.ipynb* (root)
- *Atelier-Cdiscount-pyspark.ipynb* (pyspark)
- *Atelier-Cdiscount-pyspark-pipeline.ipynb* (pyspark)

On définit une fonction de nettoyage :

```
def clean_txt(txt):
    ### remove html stuff
    txt = BeautifulSoup(txt, "html.parser", from_encoding='utf-8').get_text()
    ### lower case
    txt = txt.lower()
    ### special escaping character '...'
    txt = txt.replace(u'\u2026', '.')
    txt = txt.replace(u'\u00a0', ' ')
    ### remove accent btw
    txt = unicodedata.normalize('NFD', txt).encode('ascii', 'ignore').decode("utf-8")
    ### remove non alphanumeric char
    txt = re.sub('[^a-z_]', ' ', txt)
    ### remove french stop words
    tokens = [w for w in txt.split() if (len(w)>2) and (w not in stopwords)]
    ### french stemming
    tokens = [stemmer.stem(token) for token in tokens]
    ### tokens = stemmer.stemWords(tokens)
    return ' '.join(tokens)
```

que l'on va ensuite appliquer sur chaque description de texte.

Spark possède des TRANSFORMER qui permettent d'appliquer ces transformations.

```
STOPWORDS = set(nltk.corpus.stopwords.words('french'))
# Tokenizer
regexTokenizer = RegexTokenizer(inputCol="description", outputCol="tokenizedDescr", pattern="^[a-z_]",
                                minTokenLength=3, gaps=True)
dataTokenized = regexTokenizer.transform(dataEchDF)

# StopWordsRemover q
remover = StopWordsRemover(inputCol="tokenizedDescr", outputCol="tokenizedRemovedDescr",
stopWords = list(STOPWORDS))
dataTokenizedRemoved = remover.transform(dataTokenized)

# Stemmer
STEMMER = nltk.stem.SnowballStemmer('french')

def clean_text(tokens):
    tokens_stem = [ STEMMER.stem(token) for token in tokens]
    return tokens_stem
udfCleanText = udf(lambda lt : clean_text(lt), ArrayType(StringType()))
dataClean = dataTokenizedRemoved.withColumn("cleanDescr", udfCleanText(col('tokenizedRemovedDescr')))
```

- RegexTokenizer = Regex + Tokenizer.
- Pas de TRANSFORMER de stemming.



Fonctions de *hash* et de *tf-idf* séparées.

```
def vectorizer_train(df, columns=['Description', 'Libelle', 'Marque'], nb_hash=10000, stop_words=None):

    #HASH
    # La fonction de FeatureHasher prend en compte le nombre d'apparition de chaque mot.
    df_text = map(lambda x : collections.Counter(" ".join(x).split(" ")), df[columns].values)
    feathash = FeatureHasher(nb_hash)
    data_hash = feathash.fit_transform(map(collections.Counter, df_text))

    # TFIDF
    vec = TfidfVectorizer(min_df = 1, stop_words = stop_words, smooth_idf=True, norm='l2',
        sublinear_tf=True, use_idf=True, ngram_range=(1,2)) #bi-grams
    tfidf = vec.fit_transform(data_hash)
    return vec, feathash, tfidf


def apply_vectorizer(df, vec, columns=['Description', 'Libelle', 'Marque'], feathash):

    df_text = map(lambda x : collections.Counter(" ".join(x).split(" ")), df[columns].values)
    data_hash = feathash.transform(df_text)

    # TFIDF
    tfidf=vec.transform(data_hash)
    return tfidf
```

Fonctions de *hash* et de *tf* sont combinées dans un TRANSFORMER, et la fonction *idf* dans un second.

```
# Term Frequency
hashing_tf = HashingTF(inputCol="cleanDescr", outputCol='tf', numFeatures=10000)
trainTfDF = hashing_tf.transform(trainDF)

# Inverse Document Frequency
idf = IDF(inputCol=hashing_tf.getOutputCol(), outputCol="tfidf")
idf_model = idf.fit(trainTfDF)
trainTfIdfDF = idf_model.transform(trainTfDF)

# application à l'échantillon test
testTfDF = hashing_tf.transform(testDF)
testTfIdfDF = idf_model.transform(testTfDF)
```

## Possibilité de combiner les étapes dans un PIPELINE.

```
# Regex + Tokenizer
regexTokenizer = RegexTokenizer(inputCol="description", outputCol="tokenizedDescr",
pattern="[^a-z_]", minTokenLength=3, gaps=True)

# StopWord
remover = StopWordsRemover(inputCol="tokenizedDescr", outputCol="stopTokenizedDescr",
stopWords = list(STOPWORDS))

# Stemmer
stemmer = MyNltkStemmer(inputCol="stopTokenizedDescr", outputCol="cleanDescr")

# Indexer
indexer = StringIndexer(inputCol="categorie1", outputCol="categoryIndex")

# Hasing
hashing_tf = HashingTF(inputCol="cleanDescr", outputCol='tf', numFeatures=10000)

# Inverse Document Frequency
idf = IDF(inputCol=hashing_tf.getOutputCol(), outputCol="tfidf")

# Logistic Regression
lr = LogisticRegression(maxIter=100, regParam=0.01, fitIntercept=False, family = "multinomial",
tol=0.0001, elasticNetParam=0.0, featuresCol="tfidf", labelCol="categoryIndex")

# Creation du pipeline
pipeline = Pipeline(stages=[regexTokenizer, remover, stemmer, indexer, hashing_tf, idf, lr ])

# Execution
model = pipeline.fit(dataTrain)
```

## Définition d'un TRANSFORMER personnalisé.

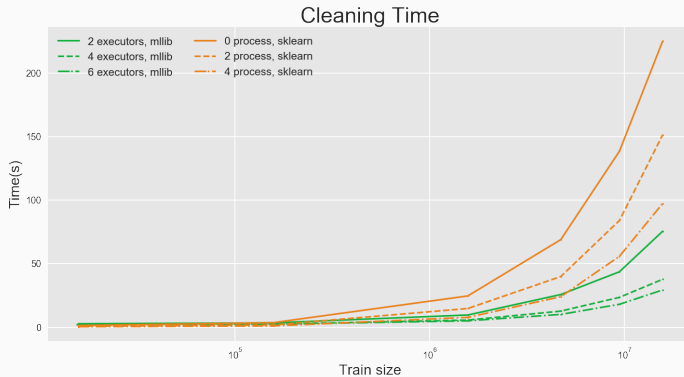
```
class MyNltkStemmer(Transformer, HasInputCol, HasOutputCol):

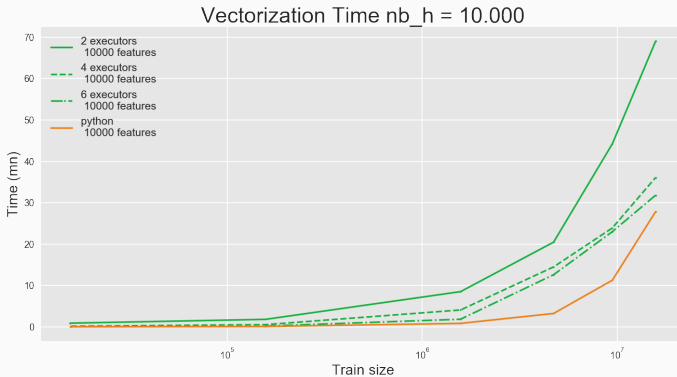
    @keyword_only
    def __init__(self, inputCol=None, outputCol=None):
        super(MyNltkStemmer, self).__init__()
        kwargs = self._input_kwargs
        self.setParams(**kwargs)

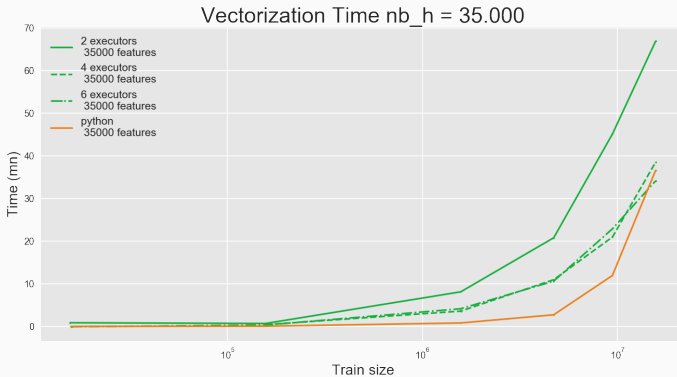
    @keyword_only
    def setParams(self, inputCol=None, outputCol=None):
        kwargs = self._input_kwargs
        return self._set(**kwargs)

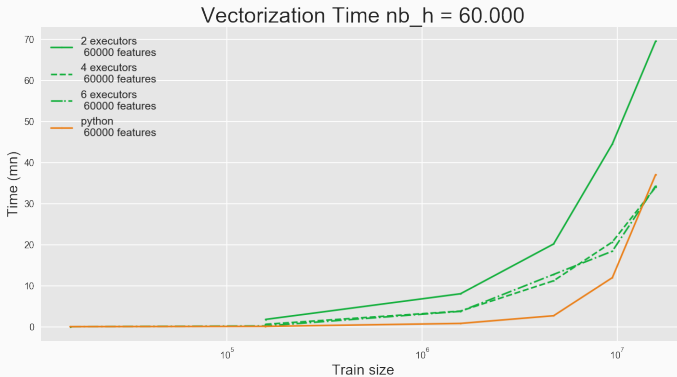
    def _transform(self, dataset):
        STEMMER = nltk.stem.SnowballStemmer('french')
        def clean_text(tokens):
            tokens_stem = [ STEMMER.stem(token) for token in tokens]
            return tokens_stem
        udfCleanText = udf(lambda lt : clean_text(lt), ArrayType(StringType()))
        out_col = self.getOutputCol()
        in_col = dataset[self.getInputCol()]
        return dataset.withColumn(out_col, udfCleanText(in_col))
```

# TP - RÉSULTAT - TEMPS NETTOYAGE



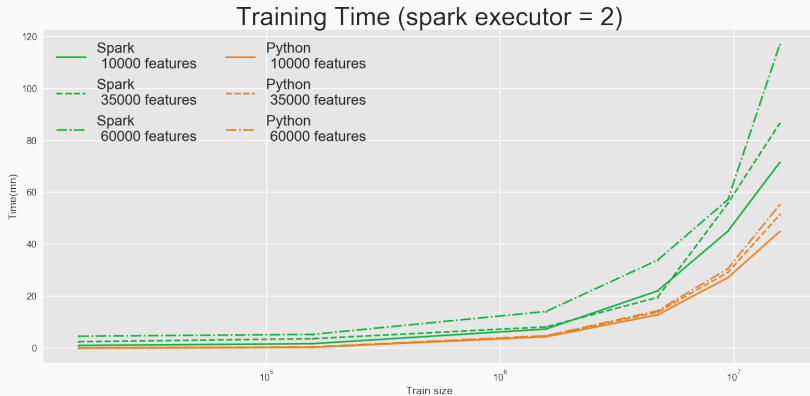




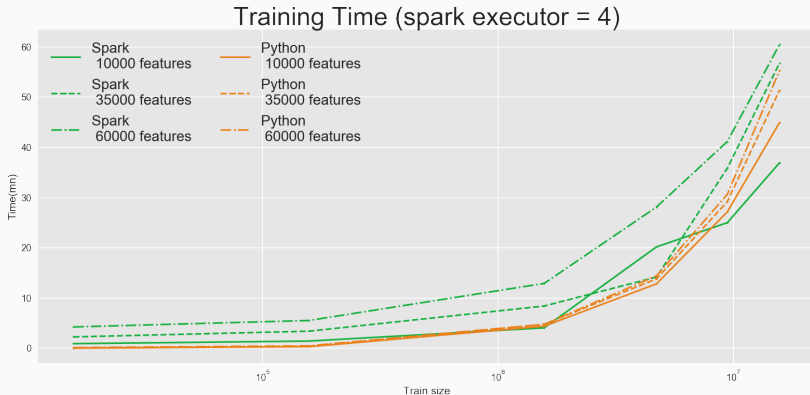




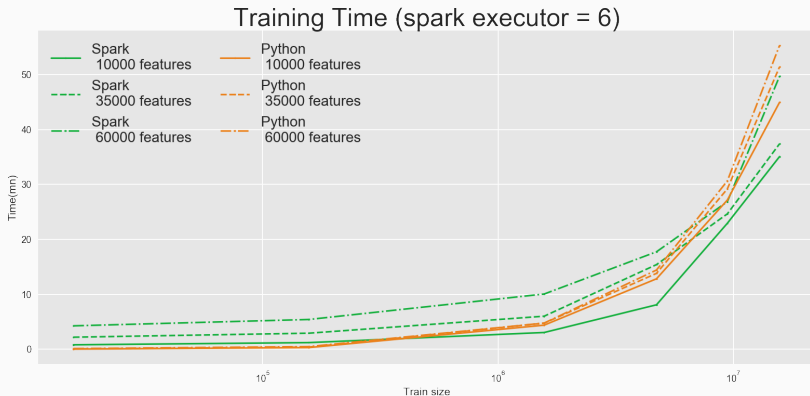
# TP - RÉSULTAT - TEMPS VECTORIZATION



# TP - RÉSULTAT - TEMPS VECTORIZATION



# TP - RÉSULTAT - TEMPS VECTORIZATION



# TP - RÉSULTAT - ERREUR PRÉDICTION



### RÉFÉRENCES

---

Bruno Goutorbe, Yang Jiao, Matthieu Cornec, Christelle Grauer, and Jérémie Jakubowicz. A large e-commerce data set released to benchmark categorization methods. 2016.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.