Université de Montpellier

Faculté des sciences

Master Informatique parcours IPS

---


HMIN326 : Fouilles de données

# Classification de documents par opinons

---

Groupe AA :

- KACIOUI Arezki
- KHEFFACHE Cherif
- SHIRALI POUR Amir

Encadrement :

- DINO Ienco
- PONCELET Pascal
- TODOROV Konstantin

Année universitaire : 2020/2021

# Sommaire

# Introduction

La fouille de données (Data Mining) est une science qui a pour objet l'extraction de connaissances à partir de grandes quantités de données par des méthodes automatiques ou semi-automatiques.

Pour ce faire, un ensemble d'algorithmes, issu de disciplines diverses tels que les statistiques, l'intelligence artificielle et des mathématiques, est utilisé afin de construire un modèle à partir des données d'entrée.

La fouille de données, est de nos jours, utilisée dans divers domaines avec des objectifs distincts allant de la gestion de la relation client à la maintenance préventive.

# Objectif

Le but de ce projet consiste à mettre en œuvre et évaluer des méthodes de classification de documents par opinion à partir d'un jeu de données comportant deux fichiers csv : **dataset.csv**, un ensemble de commentaires sur des films et **labels.csv**, la polarité des commentaires (1 : positif, -1 : négatif).

# Technologies utilisées

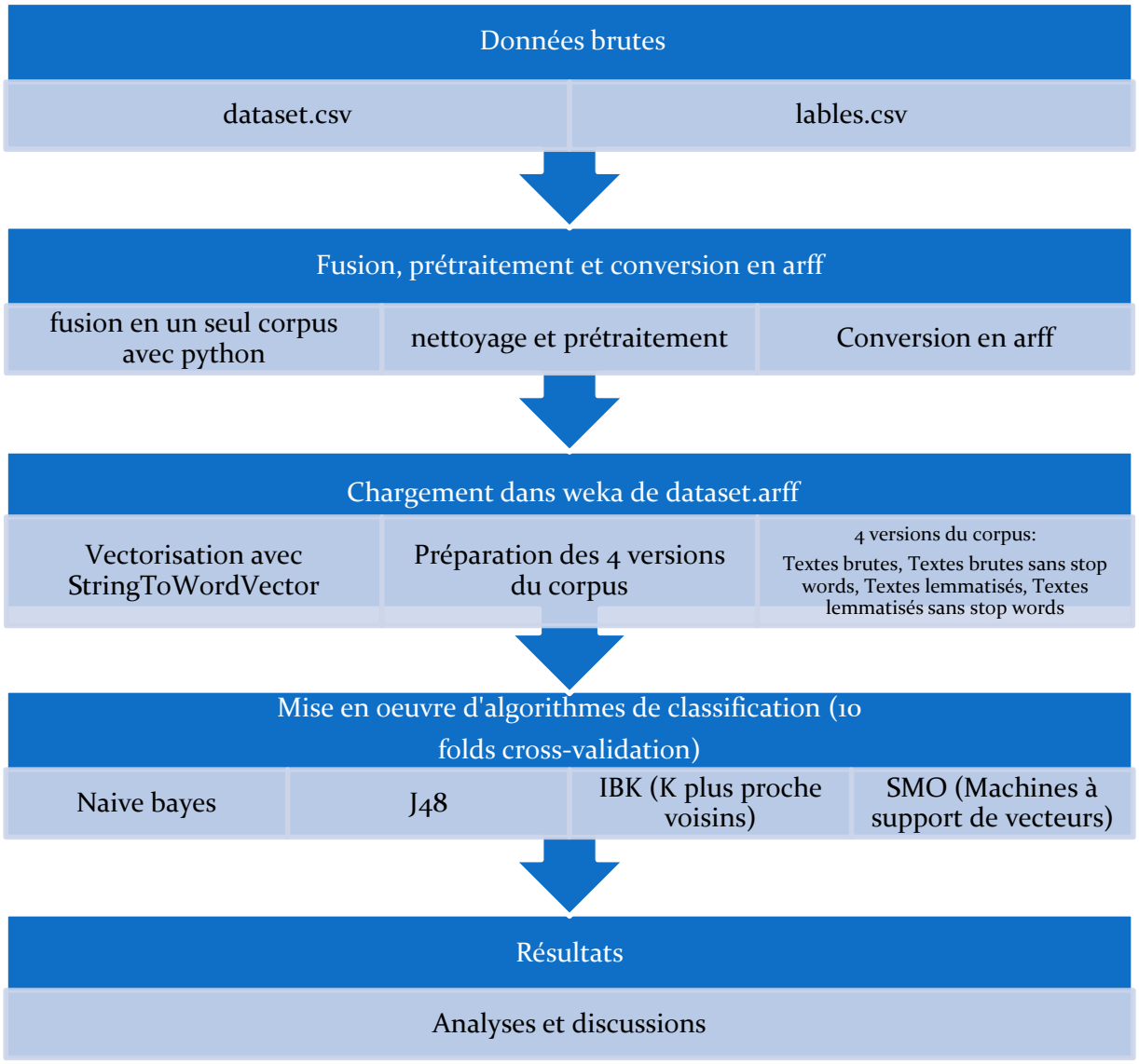
Les technologies suivantes ont été utilisées afin d'effectuer ce travail :

- **Environnement Jupyter notebook** et **Python**: ces outils nous ont servi au prétraitement des fichiers de données (suppression de la ponctuation, contractions,...) ainsi que la transformation des données en fichiers de type **.arff** pris en charge par WEKA.
- **WEKA** est une suite de logiciels d'apprentissage automatique libre et gratuite écrite en Java. Elle offre une collection d'outils de prétraitement, de classification, de clusterisation, de visualisation et d'algorithmes, pour la fouille de données et de la modélisation prédictive.
- **Fichier .arff** (Attribut Relation File Format) est un fichier texte ASCII qui décrit une liste d'instances partageant un ensemble d'attributs. Les fichiers ARFF ont été développés par le Machine Learning Project du Département d'informatique de l'Université de Waikato pour être utilisés avec le logiciel d'apprentissage automatique Weka.[1]

[1] www.cs.waikato.ac.nz

# Méthodologie

Nous avons suivi les étapes résumées dans le schéma ci-dessous afin de réaliser ce projet :




## Fusion, prétraitement et conversion en ARFF

Dans cette étape, nous avons utilisé l'environnement Jupyter :

- Import des données csv (dataset.csv et lables.csv) avec pandas.
- Fusion des deux DataFrame.
- Prétraitement avec Python (suppression des caractères non ASCII, des nombres, des contractions et de la ponctuation)
- Récupération de la liste des stop-words avec le module NLTK dans un fichier stopWords.txt
- Transformation du DataFrame en .arff avec le module arff => dataset.arff.

## Prétraitement avec WEKA

Dans cette partie du projet, nous avons effectué la vectorisation et les différents filtres nécessaires afin d'avoir les différents types de données adaptés à Weka.

La méthode de pondération fréquentielle (TF-IDF : Term Frequency-Inverse Document Frequncy) à l'aide de la fonction StringToWordVector de weka a été utilisé pour ce faire.

Nous avons obtenu à la fin, quatre corpus différents :

- Textes brutes (1-dataset_raw.arff) : il s'agit du fichier dataset.arff auquel on a appliqué la fonction StringToWordVector.
- Textes lemmatisés (2-dataset_lemmatized.arff) : il s'agit du fichier dataset.arff auquel on a appliqué la fonction StringToWordVector avec le filtre IteratedLovinsStemmer.
- Textes brutes sans stop-words (3-dataset_raw_withoutStopWords.arff) : il s'agit du fichier dataset.arff auquel on a appliqué la finction StringToWordVector avec le filtre stopWordsHandler ayant comme paramètre le fichier stopWords.txt précédemment sauvé.
- Textes lemmatizé sans stop-words (4-datasetlemmatized withoutStopWords.arff) : il s'agit du fichier dataset.arff auquel on a appliqué la fonction StringToWordVector avec le filtre IteratedLovinsStemmer et le filtre stopWordsHandler ayant comme paramètre le fichier stopWords.txt.

On a également effectué les mêmes opérations de prétraitement directement avec Python, mais vu le peu de différences observés sur les résultats, on a opté pour Weka.

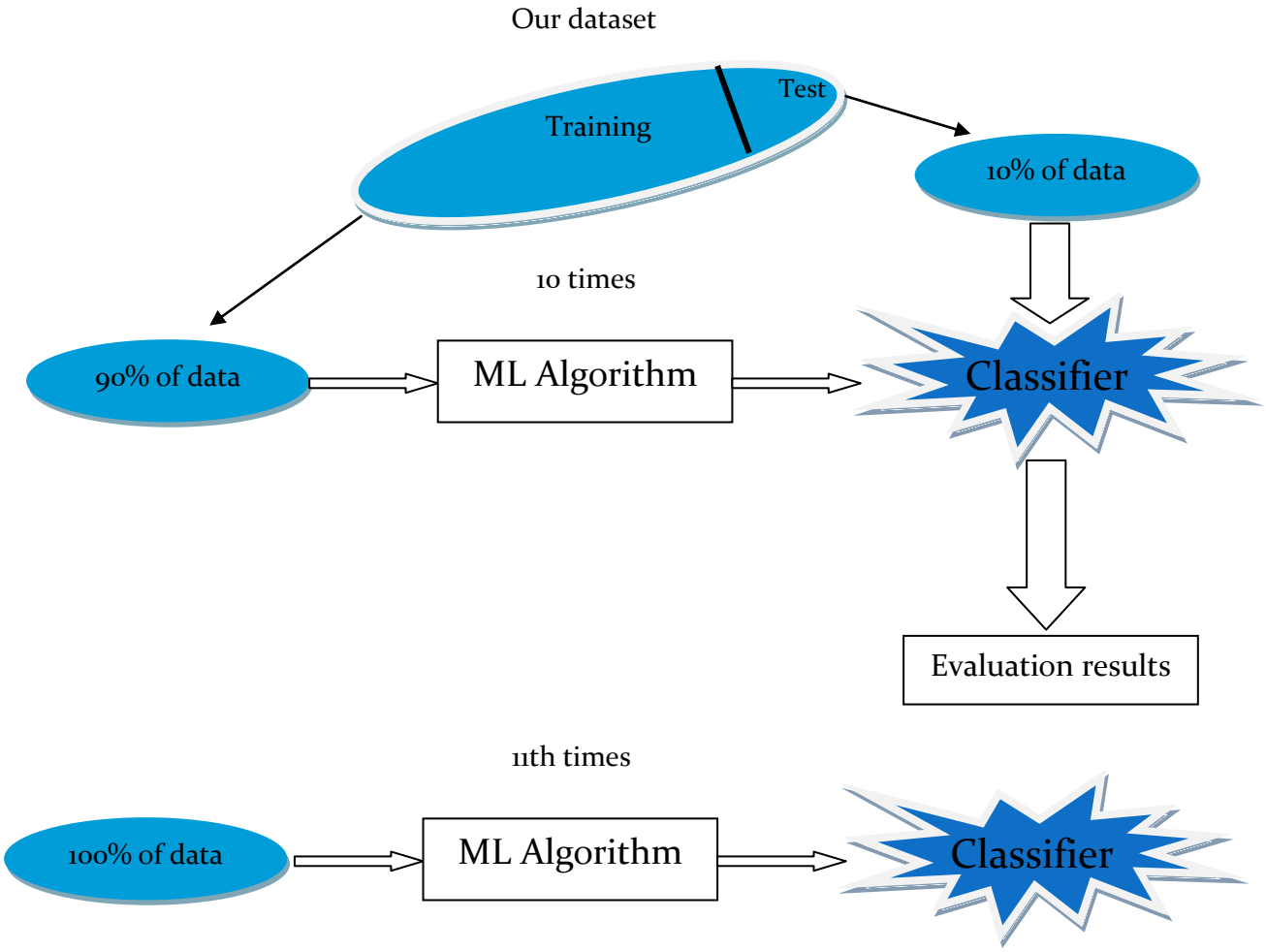
## Mise en œuvre d'algorithmes de classification

Dans cette partie, nous avons effectué quelques tests avec différentes options à savoir use training test et supplied test set, Percentage split avec 80% et Cross-validation Folds 10 sur différents classificateurs et corpus avant d'opter pour l'option finale.

L'option retenue est le Cross-validation avec un fold de dix, car elle permet le découpage du fichier entier en dix blocs, neuf étant réservé a l'apprentissage et un bloc aux tests.

Cet ensemble est ensuite analysé par les algorithmes de classification dix fois, minimisant ainsi la variance.

Le fonctionnement du 10 folds cross-validation est expliqué dans la figure ci-dessous.




**10 folds cross-validation**

### Naive Bayes

Un classificateur naïf bayésien un type de classificateur probabiliste basé sur le théorème de Bayes[2]. Il est très populaire en Machine Learning. C'est un algorithme d'apprentissage supervisé[3], particulièrement utile pour les problématiques de classification de texte.

Il est considéré comme naïf, car il suppose que l'existence d'une caractéristique pour une classe est indépendante de l'existence d'autre caractéristique. Il produit un model pour chaque classe.

[2] https://fr.wikipedia.org/wiki/Théorème_de_Bayes
[3] https://en.wikipedia.org/wiki/Supervised_learning

### K plus proche voisin : IBK

L'algorithme IBK est une implémentation dans Weka de la méthode des k plus proches voisins.

C'est une méthode d'apprentissage supervisée pouvant être utilisée aussi bien pour la classification que pour la régression. Elle permet de prédire la classe d'une instance selon la classe majoritaire de ses « k » voisins.

Le K-NN est catégorisé dans le Lazy Learning[4] car il n'a pas besoin de construire un modèle prédictif. Il se base uniquement sur les jeux de données pour prédire un résultat.

### Arbres de décision : J48

J48 est une implémentation dans Weka de l'algorithme C4.5, une amélioration de l'algorithme ID3[5]. Cet algorithme permet de générer un arbre de décision afin de faire de la classification.

Le meilleur attribut de l'ensemble est placé à la racine de l'arbre. Par la suite le jeu d'apprentissage est devisé en sous-ensemble de manière à ce que chaqu'un contienne des données équivalentes pour un attribut.

### Algorithme SMO

Le SMO est une implémentation de l'algorithme d'optimisation séquentielle minimale pour former un classificateur de vecteur de support (SVC).

Cette implémentation remplace globalement toutes les valeurs manquantes et transforme les attributs nominaux en attributs binaires. Il normalise également tous les attributs par défaut. (Dans ce cas, les coefficients de la sortie sont basés sur les données normalisées, pas sur les données d'origine). Les problèmes multi-classes sont résolus en utilisant la classification par paires (alias 1 contre 1).

[4] https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-30164-8_443
[5] https://fr.wikipedia.org/wiki/Algorithme_ID3

# Résultats et discussion

Tous les fichiers d'entrés ont un total de 10000 instances avec 5000 avis positif et 5000 avis négatif.

A l'issu des traitements, les résultats suivants ont été obtenus.

## Selon le classificateur

### Résultats obtenus avec Naïve bayes

| Type de corpus | Précision | Rappel | Corrects | Incorrects |
|---|---|---|---|---|
| Textes brutes | 0.858 | 0.858 | 85.77 % | 14.23 % |
| Textes lemmatisés | 0,854 | 0,850 | 85.04 % | 14.96 % |
| Textes bruts sans stop-words | 0.876 | 0.876 | 87.59 % | 12.41 % |
| Textes lemmatisés sans stop-words | 0.856 | 0.853 | 85.27 % | 14.73 % |

La classification avec cet algorithme est la plus performante en termes de temps d'exécution.

On constate que le Naïve Bayes a enregistré la meilleure précision (0.876) avec les Textes brutes sans stop-words.

Néanmoins la différence de précision reste minime entre les quatre différents corpus (+/- 0.02). Cela signifie que les prétraitements n'ont pas apporté plus de précision.

### Résultats obtenus avec K plus proche voisin : IBK (K = 1)

| Type de corpus | Précision | Rappel | Corrects | Incorrects |
|---|---|---|---|---|
| Textes brutes | 0.632 | 0.631 | 63.14 % | 36.86 % |
| Textes lemmatisés | 0,631 | 0,631 | 63.06 % | 36.94 % |
| Textes bruts sans stop-words | 0.597 | 0.594 | 59.39 % | 40.61 % |
| Textes lemmatisés sans stop-words | 0,604 | 0,594 | 59.35 % | 40.65 % |

L'IBK est celui qui a obtenu les moins bons résultats en termes de précision avec des temps d'exécutions moyens. On constate par ailleurs que la suppression des mots vides à un effet négatif sur la précision de la méthode (+/- 0.030).

### Résultats obtenus avec l'arbre de décision : J48

| Type de corpus | Précision | Rappel | Corrects | Incorrects |
|---|---|---|---|---|
| Textes brutes | 0.755 | 0.755 | 75.51 % | 24.49 % |
| Textes lemmatisés | 0,758 | 0,758 | 75.78 % | 24.22 % |
| Textes bruts sans stop-words | 0.764 | 0.764 | 76.37 % | 23.63 % |
| Textes lemmatisés sans stop-words | 0.760 | 0.760 | 76.03 % | 23.97 % |

L'analyse de ces résultats ne dévoile aucune différence significative hormis une légère amélioration lors du traitement sans stop-words. On en déduit que les prétraitements n'impactent pas considérablement la précision.

Par ailleurs, les temps d'exécutions restent assez importants en comparaison avec les autres classificateurs.

### Résultats obtenus avec l'algorithme SMO

| Type de corpus | Précision | Rappel | Corrects | Incorrects |
|---|---|---|---|---|
| Textes brutes | 0.869 | 0.869 | 86.94 % | 13.06 % |
| Textes lemmatisés | 0,871 | 0,871 | 87.08 % | 12.92 % |
| Textes bruts sans stop-words | 0.867 | 0.867 | 86.69 % | 13.31 % |
| Textes lemmatisés sans stop-words | 0,873 | 0,873 | 87.26 % | 12.74 % |

Cet algorithme est mieux noté en matière de précision sur l'ensemble des corpus. Par ailleurs, il enregistre des temps de traitements assez longs.

On observe, en outre, que les deux modèles lemmatisés se démarquent avec de meilleures précisions, respectivement, 0.873 et 0.871 pour les textes lemmatisés sans stop-words et les textes lemmatisés.

Cette analyse met en évidence l'influence de la lemmatisation sur le traitement.

## Selon le modèle

### Résultats obtenus avec  les textes brutes

| Type de corpus | Précision | Rappel | Corrects | Incorrects |
|---|---|---|---|---|
| SMO | 0.869 | 0.869 | 86.94 % | 13.06 % |
| IBK | 0.632 | 0.631 | 63.14 % | 36.86 % |
| Naive Bayes | 0.858 | 0.858 | 85.77 % | 14.23 % |
| J48 | 0.755 | 0.755 | 75.51 % | 24.49 % |
| Random forest | 0.861 | 0.861 | 86.09 % | 13.91 % |
| Naive Bayes Multinominal | 0.860 | 0.860 | 86.02 % | 13.98 % |

On note que pour ce type de corpus, le SMO réalise la meilleure précision (0.869).

### Résultats obtenus avec les textes lemmatisés

| Type de corpus | Précision | Rappel | Corrects | Incorrects |
|---|---|---|---|---|
| SMO | 0,871 | 0,871 | 87.08 % | 12.92 % |
| IBK | 0,631 | 0,631 | 63.06 % | 36.94 % |
| Naive Bayes | 0,854 | 0,850 | 85.04 % | 14.96 % |
| J48 | 0,758 | 0,758 | 75.78 % | 24.22 % |

On note que pour ce type de corpus, le SMO réalise la meilleure précision (0.871).

### Résultats obtenus avec les textes brutes sans Stop-words

| Type de corpus | Précision | Rappel | Corrects | Incorrects |
|---|---|---|---|---|
| SMO | 0.867 | 0.867 | 86.69 % | 13.31 % |
| IBK | 0.597 | 0.594 | 59.39 % | 40.61 % |
| Naive Bayes | 0.876 | 0.876 | 87.59 % | 12.41 % |
| J48 | 0.764 | 0.764 | 76.37 % | 23.63 % |

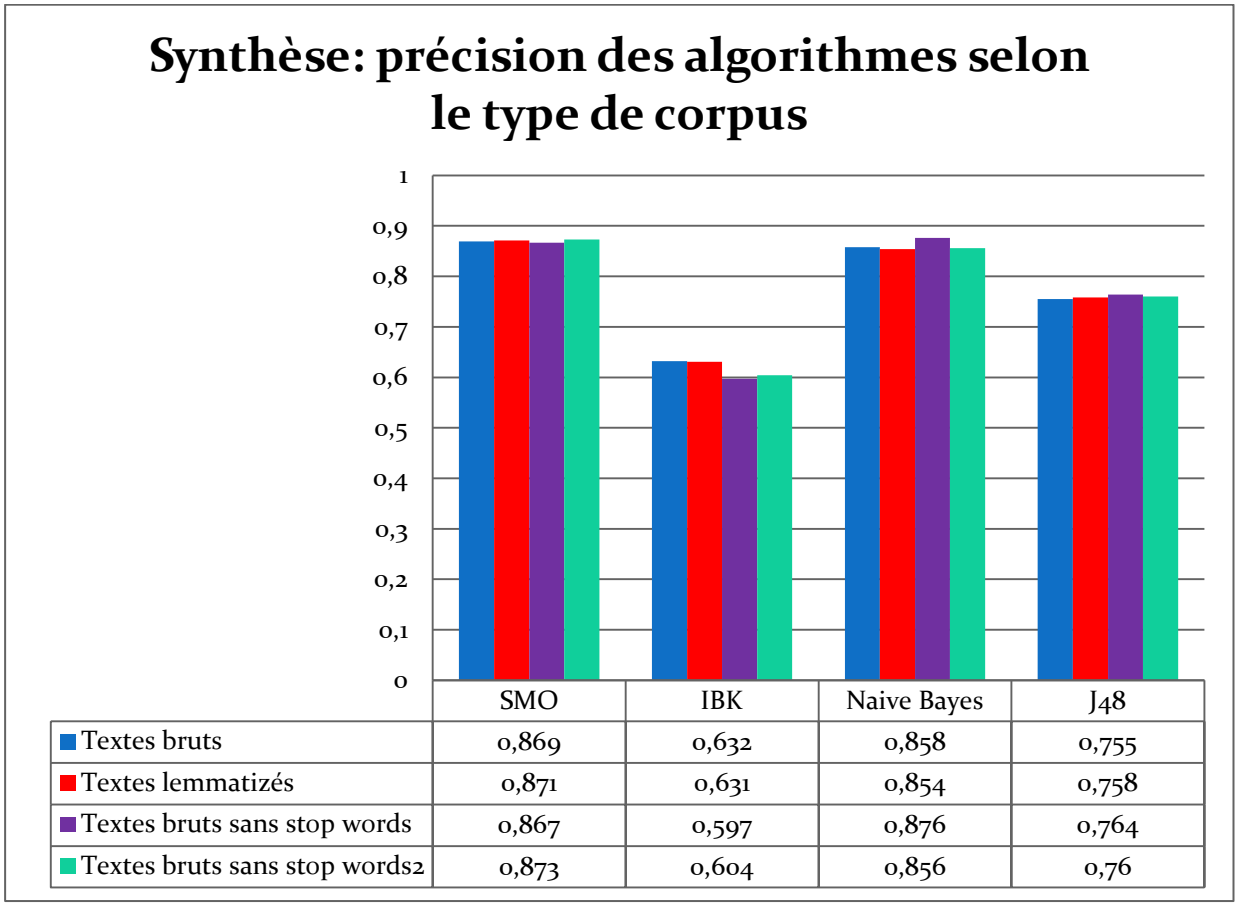
On note que pour ce type de corpus, le Naïve Bayes réalise la meilleure précision (0.876).

### Résultats obtenus avec les textes lemmatisés sans stop-words

| Type de corpus | Précision | Rappel | Corrects | Incorrects |
|---|---|---|---|---|
| SMO | 0,873 | 0,873 | 87.26 % | 12.74 % |
| IBK | 0,604 | 0,594 | 59.35 % | 40.65 % |
| Naive Bayes | 0.856 | 0.853 | 85.27 % | 14.73 % |
| J48 | 0.760 | 0.760 | 76.03 % | 23.97 % |

On note que pour ce type de corpus, le SMO réalise la meilleure précision (0.873).

## Synthèse




| | SMO | IBK | Naive Bayes | J48 |
|---|---|---|---|---|
| Textes bruts | 0,869 | 0,632 | 0,858 | 0,755 |
| Textes lemmatizés | 0,871 | 0,631 | 0,854 | 0,758 |
| Textes bruts sans stop words | 0,867 | 0,597 | 0,876 | 0,764 |
| Textes bruts sans stop words2 | 0,873 | 0,604 | 0,856 | 0,76 |

On observe à la suite de ces testes que l'algorithme SMO est celui qui donne les meilleurs performances générale de classification, notamment avec les corpus lemmatisés, suivi de très prés par le Naïve Bayes.

Le K plus proche voisin (IBK) est celui qui enregistre le résultat le moins performant avec une précision de moins de 63% pour tous les fichiers.

Selon l'algorithme, le prétraitement peut avoir plus au moins d'effet.

Le SMO et le Naïve Bayes se démarquent quant à l'IBK et le J48 avec une légère performance  pour le SMO. Par ailleurs, vu les temps d'analyses et la proximité des résultats nous pensons que le Naïve Bayes est le plus adapté pour de la classification sur ce set de données.

# Conclusion

Ce travail nous a permis de nous initier à la fouille de données, à quelques algorithmes utilisés dans ce sens, de saisir l'impact que peut avoir un prétraitement différent sur une prédiction et l'importance du choix adéquat de la méthodologie de traitement selon la nature des données.

En perspective, le même travaille pourra être réalisé avec d'autres outils, notamment Scikit-learn, Tree-tagger, d'autres algorithmes et un set de données plus complexes afin de comparer les résultats et la performance des outils.