Apprentissage automatique

# Utilisation de la librairie scikit-learn pour la classification des fake news

Abusalha Y., Bampatzani S., Bellato S., Dehareng M.

M2 TAL IM

Institut National des Langues et Civilisations Orientales

# Table des matières

# Partie I : problématique et corpus

## 1. Corpus

Lors de ce projet, notre but était de tester différents algorithmes de classification avec la librairie d'apprentissage automatique Scikit-Learn, afin de classer des articles de presse en tant que "fake" ou "trusted".

Notre corpus se compose de 148 articles de presse répartis en un corpus train "*all-train.xml*" contenant 99 articles (42 "fake", 37 "trusted" et 20 "parodic") et un corpus test "*all-test.xml*" contenant 49 articles (16 "fake", 17 "trusted" et 16 "parodic").

## 2. Problématique

Pour construire un bon modèle d'apprentissage, il est nécessaire de minimiser les erreurs sur la base d'apprentissage et de construire un système capable de généraliser correctement. Pour ce faire, nous avons utilisé la librairie d'apprentissage automatique Scikit-Learn.

Notre but étant la classification des fake news, nous avons implémenté plusieurs algorithmes d'apprentissage afin de trouver celui qui était le plus adapté à notre problématique. Parmi ces différents algorithmes d'apprentissage, nous avons utilisé les suivants : les K plus proches voisins, TF.IDF, Naïve Bayes, les machines à vecteurs support, les arbres de décision et un perceptron.

# Partie II : analyse

## 1. SVM

Les SVM (*Support Vector Machines*, ou *machines à vecteurs support* en français) ont plusieurs avantages théoriques et pratiques, devenant ainsi un outil très puissant pour résoudre plusieurs problèmes de classification.

Le classificateur SVM appartient à la catégorie des classifieurs linéaires. Cela implique qu'il implémente une séparation linéaire des données. Afin de pouvoir effectuer cette séparation, le SVM a besoin de données d'entraînement, autrement dit, un training set. Dans notre cas, le corpus d'entraînement fourni est le fichier *all-train.xml*. A partir de ces données, le SVM estime la meilleure séparation plausible. Après cela, il est capable de prédire à quelle catégorie appartient une nouvelle entrée sans intervention humaine. Cela se fait lorsque nous donnons au modèle le fichier *all-test.xml*. Les prédictions du modèle (*model*) se trouvent dans le fichier *output.xml*.

Le résultat obtenu avec un kernel *linear* est un score de 0.387. En effet, ce modèle d'apprentissage automatique est conçu de telle manière que la séparation des données est forcément une ligne droite. Pour notre jeu de données, c'est loin d'être suffisant. Pour cela, nous utilisons un kernel *rbf* (*Radial Basis Function*) et nous arrivons à obtenir un score légèrement élevé de 0.428. Malgré plusieurs modifications des paramètres *C* et *gamma*, nous n'avons pas réussi à augmenter ce score.

## 2. Naïve Bayes

Un autre modèle adapté à la classification de documents, le modèle de Naïve Bayes. Il s'agit d'un algorithme d'apprentissage automatique supervisé, basé sur l'application du théorème bayésien, qui suppose « naïvement » que les hypothèses, les *features*, sont indépendantes.

Pour implémenter cet algorithme, il est nécessaire de choisir a priori une distribution pour les classes et pour les caractéristiques. La librairie Scikit-Learn propose trois distributions, dont *GaussianNB*, *MultinomialNB* et enfin *BernouilliNB*.

Le classifieur *GaussianNB* est basé sur une distribution de probabilité de type Gaussienne, autrement dit, normale. Ce classifier est plus adapté pour des valeurs continues. Notre jeu de données ne contenant pas des valeurs continues, il est logique que nous obtenions un score de 0.38.

En ce qui concerne le modèle *BernouilliNB*, il est implémenté pour des caractéristiques binaires, et conséquemment, la fréquence des mots n'importe pas, non plus que leur ordre d'apparition. Le faible score que nous obtenons, qui est environ 0.43, n'est encore une fois pas surprenant grâce à la nature de nos corpus.

Enfin, le modèle *MultinomialNB* est le plus adapté à notre jeu de données, car la loi multinomiale est plus adaptée pour des valeurs discrètes. Ce modèle multinomial construit un vecteur constitué des indices des mots dans un dictionnaire. De cette manière, il conserve l'ordre des mots du document à classifier. En général, entre les trois modèles, il donne de meilleurs résultats. Cela est affirmé pour notre jeu de données, le score obtenu étant environ 0.53.

## 3. KNN

L'algorithme des "k-plus proches voisins" est un algorithme simple dans son principe et qui est assez efficace dans certains contextes. Le principe de cet algorithme est de chercher, pour chaque donnée du corpus *test*, les k données annotées du corpus *train* dont la valeur est la plus "proche" (au sens d'une distance prédéfinie) de cette donnée *test* et d'associer à cette dernière la classe majoritaire parmi ces k données *train* les plus proches. L'algorithme impose donc de calculer la distance entre la donnée test et toutes celles fournies en exemples et de mémoriser la classe des k plus proches. L'algorithme nécessite donc aussi de paramétrer k, le nombre de voisins à considérer. Concernant la distance entre deux textes, plusieurs calculs peuvent être considérés, comme par exemple la distance de Manhattan, la distance Euclidienne, la distance de Jaccard ou encore la similarité cosinus.

Le meilleur score obtenu avec cet algorithme grâce à Scikit-Learn est de 0.45. Ce score a été obtenu en utilisant la distance Euclidienne (par rapport à la distance de Manhattan qu'il est aussi possible d'implémenter avec scikit-learn) et un nombre de K plus proche voisin égal à 4.

| | Trusted | Fake | Parodic |
|---|---|---|---|
| Trusted | 9 | 5 | 3 |
| Fake | 8 | 4 | 4 |
| Parodic | 4 | 3 | 9 |

Tableau 1 Matrice de confusion du KNN

Avec cette matrice de confusion nous pouvons voir que :

- La classe *trusted* n'est bien repérée que la moitié du temps.
- La classe *fake* est la classe la moins bien repérée, elle est souvent confondue avec la catégorie *trusted*.
- La classe *parodic* est confondue plus souvent avec la catégorie *trusted* qu'avec la catégorie *fake* qui s'en rapprocherait pourtant plus au niveau du sens.
- La majorité des documents a été classée dans la catégorie *trusted*.

## 4. Perceptron

Le perceptron est un algorithme d'apprentissage supervisé de classifieurs binaires (c'est-à-dire séparant deux classes, ici *trusted* et *fake*). Le perceptron est le réseau de neurone le plus simple car il ne contient qu'une seule couche. C'est un classifieur linéaire.

Sachant que le perceptron est un algorithme binaire, afin de ne pas fausser les résultats en classant les documents parodiques comme étant fake dès le départ, nous avons décidé de ne pas prendre en compte les documents parodiques du corpus train lors de la création du modèle.

Ce modèle donne un score d'environ 0.33.

| | Trusted | Fake |
|---|---|---|
| Trusted | 0 | 17 |
| Fake | 0 | 16 |
| Parodic | 0 | 16 |

Tableau 2 Matrice de confusion du Perceptron

Avec cette matrice de confusion nous pouvons voir que :

- Tous les résultats ont été classés en tant que *fake*, le perceptron a donc été "activé" pour tous les documents du corpus test. Ce modèle n'est donc pas forcément le meilleur pour catégoriser notre corpus avec les features que nous avions définis.

## 5. Arbre de décision

Un arbre de décision (AD) est un arbre binaire, c'est-à-dire un arbre dont les nœuds ont au plus deux fils. Chaque nœud correspond à un test effectué sur une valeur xi d'une feature représentée par un vecteur x. Une fois l'arbre construit, classifier un nouvel candidat se fait par une descente dans l'arbre, de la racine vers un des nœuds.

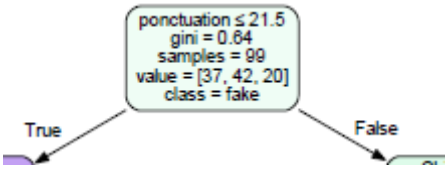


Figure 1 nœud racine de l'arbre

Ce modèle est relativement facile à implémenter étant donné qu'il ne nécessite pas de paramétrage. En revanche, il est relativement dépendant de la qualité des features et doit donc être traité en conséquence. Ainsi, pour notre problématique, nous obtenons un score de 0.45. Pour évaluer l'impact du nombre de classes à prédire sur l'efficacité de ce modèle, nous avons supprimé la classe 'parodic' dans les corpus d'entraînement et de test. Cette expérience porte le score à 0.61. Les tableaux suivants présentent les statistiques détaillées des deux expériences.

| | trusted | fake |
|---|---|---|
| trusted | 9 | 8 |
| fake | 5 | 11 |

Tableau 3 Matrice de confusion (2 classes)

| | précision | rappel | f-mesure |
|---|---|---|---|
| trusted | 0.64 | 0.53 | 0.58 |
| fake | 0.58 | 0.69 | 0.63 |
| total | 0.61 | 0.61 | 0.60 |

Tableau 4 évaluation (2 classes)

| | trusted | fake | Parodic |
|---|---|---|---|
| trusted | 7 | 3 | 7 |
| fake | 2 | 9 | 5 |
| parodic | 3 | 7 | 6 |

Tableau 5 Matrice de confusion (3 classes)

| | précision | rappel | f-mesure |
|---|---|---|---|
| trusted | 0.58 | 0.38 | 0.35 |
| fake | 0.47 | 0.45 | 0.45 |
| parodic | 0.33 | 0.38 | 0.35 |
| total | 0.47 | 0.45 | 0.45 |

Tableau 6 évaluation (3 classes)

## 6. Features

Pour analyser et modéliser un texte, nous avons besoin de le convertir en features (représentation numérique). Nous avons donc décidé de faire nos features manuellement afin de faire nos tests et analyser les résultats. Nous avons combiné ces features avec celles obtenues grâce à la méthode statistique TF-IDF puis nous avons comparé les résultats.

Le TF-IDF est une méthode statistique qui vise à refléter l'importance d'un mot dans un document d'un corpus. Elle augmente proportionnellement selon le nombre de fois qu'un mot apparaît dans un document. Bien que le TF-IDF soit une bonne métrique pour extraire les features, elle ne prend pas en compte la position ou le contexte d'un mot.

Nous avons décidé de combiner les 17 features que nous avions déjà obtenues à l'aide de notre script getfeatures.py avec une partie des features du TF-IDF. Pour commencer, nous avons choisi 100 features mais les résultats n'étaient pas concluants. En passant à 50, les résultats se sont quelque peu améliorés. Après plusieurs tests nous avons finalement choisi 20, nombre idéal pour obtenir de bons résultats.

Nous avons utilisé la fonction (*hstack*) de Numpy pour concaténer les 17 features de base avec les 20 features de TF-IDF. Ceci nous a permis de créer des vecteurs/matrices plus grands à partir des vecteurs/matrices plus petits.

## 7. Régression logistique

Modèle de régression logistique : la RL est un modèle de statistiques et de classifications populaire dans le domaine de machine learning (souvent binaire : 2 classes).

Contrairement à ce que le nom suggère, la régression logistique est une méthode de classification qui procède à une régression sur des attributs discrets. Son avantage est qu'en plus de renvoyer l'attribut prédit, l'algorithme renvoie un indicateur de confiance relatif à la prédiction.

## 8. Analyse comparative des algorithmes

Voici un tableau des différents résultats des modèles obtenus avec les features de base puis combinées avec les features de TF-IDF (17+20 =32).

| *Features* | *KNN* | *Naïve Bayes Multinomial* | *Decision tree* | *Logistic regression* | *SVM (rbf)* |
|---|---|---|---|---|---|
| *getfeatures.py sans TF-IDF features* | 0.45 | 0.53 | 0.61 | Non | 0.43 |
| *TF-IDF features + getfeatures.py 2 classes* | 0.57 | 0.40 | 0.55 | 0.55 | 0.20 |
| *TF-IDF features + getfeatures.py 3 classes* | 0.32 | 0.32 | 0.36 | 0.34 | 0.20 |

Tableau 7 Comparaison des algorithmes

Nous pouvons constater que l'ajout des features TF-IDF n'a pas toujours un impact positif, comme dans le cas de Naïve Bayes Multinomial, de l'arbre de décision et de SVM. Par contre, cela améliore le score du KNN.

# Conclusion

Les modèles de régression logistique et l'arbre de décision ont produit les meilleurs résultats après avoir ajouté le TF-IDF aux features. Le modèle de régression logistique est à privilégier en raison de sa rapidité d'implémentation et de sa complexité en temps. Il est intéressant de mentionner le fait que le TF-IDF a eu un impact positif à deux algorithmes non adaptés à notre jeu de données ; le Naïve Bayes Gaussien et le SVM linéaire qui ont respectivement produit avec deux classes, un score de 0.67 et 0.65.

Il pourrait être très intéressant par la suite de travailler et d'améliorer les features utilisés pour pouvoir augmenter les résultats.

En ce qui concerne les avantages de la librairie Scikit-Learn, elle est très intuitive dans son implémentation. Elle permet d'implémenter une grande variété d'algorithmes d'apprentissage automatique. Par ailleurs, la documentation pour ces algorithmes est très claire.

Concernant le corpus, nous avons pu nous apercevoir que ce dernier n'était pas annoté de la même manière tout le temps. En effet, quelque fois les articles provenant de Le Figaro étaient annotés comme étant « fake » et d'autres fois ils étaient annotés comme étant « trusted ». Cela peut alors causer des problèmes. Un même article dupliqué a même une fois été annoté comme « fake » et une autre fois comme « parodic ».

# Annexes

| **Sources** | |
|---|---|
| Le Monde | www.lemonde.fr/ |
| Le Figaro | www.lefigaro.fr/ |
| Le Parisien | www.leparisien.fr/ |
| 20 Minutes | www.20minutes.fr/ |
| Libération | www.liberation.fr/ |
| Wikistrike | www.wikistrike.com/ |
| Bilboquet Magazine | www.bilboquet-magazine.fr/ |
| Réseau International | https://reseauinternational.net/ |
| La Gauche m'a tuer | http://lagauchematuer.fr/ |
| Nord Presse | http://nordpresse.be/ |
| Le Navet | http://lenavet.ca/ |
| Pire Médias | http://www.piremedias.com/ |
| Le Soir | http://www.lesoir.be/ |
| Le site d'Initiative Citoyenne | http://initiativecitoyenne.be/ |
| RadioCockpit.fr | https://www.radiocockpit.fr/ |
| Mes Propres Recherches | http://www.mespropresrecherches.com/ |
| Rispa.fr | http://www.rispa.fr/ |


ponctuation ≤ 21.5
gini = 0.64
samples = 99
value = [37, 42, 20]
class = fake
True
False
adv ≤ 0.5
gini = 0.557
samples = 25
value = [6, 4, 15]
class = parodic
Chine ≤ 0.5
gini = 0.556
samples = 74
value = [31, 38, 5]
class = fake
selon ≤ 0.5
gini = 0.444
samples = 3
value = [1, 2, 0]
class = fake
impers ≤ 0.5
gini = 0.475
samples = 22
value = [5, 2, 15]
class = parodic
vs ≤ 1.5
gini = 0.568
samples = 62
value = [31, 26, 5]
class = trusted
gini = 0.0
samples = 12
value = [0, 12, 0]
class = fake
gini = 0.0
samples = 2
value = [0, 2, 0]
class = fake
gini = 0.0
samples = 1
value = [1, 0, 0]
class = trusted
superlatif ≤ 0.5
gini = 0.58
samples = 10
value = [5, 1, 4]
class = trusted
adv ≤ 7.5
gini = 0.153
samples = 12
value = [0, 1, 11]
class = parodic
russe ≤ 0.5
gini = 0.551
samples = 50
value = [29, 16, 5]
class = trusted
suspen ≤ 1.5
gini = 0.278
samples = 12
value = [2, 10, 0]
class = fake
adv ≤ 3.5
gini = 0.449
samples = 7
value = [5, 1, 1]
class = trusted
gini = 0.0
samples = 3
value = [0, 0, 3]
class = parodic
gini = 0.0
samples = 11
value = [0, 0, 11]
class = parodic
gini = 0.0
samples = 1
value = [0, 1, 0]
class = fake
superlatif ≤ 1.5
gini = 0.531
samples = 47
value = [29, 13, 5]
class = trusted
gini = 0.0
samples = 3
value = [0, 3, 0]
class = fake
vs ≤ 8.0
gini = 0.444
samples = 6
value = [2, 4, 0]
class = fake
gini = 0.0
samples = 6
value = [0, 6, 0]
class = fake
ponctuation ≤ 11.5
gini = 0.625
samples = 4
value = [2, 1, 1]
class = trusted
gini = 0.0
samples = 3
value = [3, 0, 0]
class = trusted
impers ≤ 0.5
gini = 0.484
samples = 40
value = [27, 9, 4]
class = trusted
selon ≤ 0.5
gini = 0.571
samples = 7
value = [2, 4, 1]
class = fake
impers ≤ 4.5
gini = 0.32
samples = 5
value = [1, 4, 0]
class = fake
gini = 0.0
samples = 1
value = [1, 0, 0]
class = trusted
gini = 0.0
samples = 1
value = [0, 1, 0]
class = fake
ponctuation ≤ 16.5
gini = 0.444
samples = 3
value = [2, 0, 1]
class = trusted
adv ≤ 18.0
gini = 0.165
samples = 11
value = [10, 0, 1]
class = trusted
ponctuation ≤ 35.5
gini = 0.549
samples = 29
value = [17, 9, 3]
class = trusted
ponctuation ≤ 44.0
gini = 0.625
samples = 4
value = [2, 1, 1]
class = trusted
gini = 0.0
samples = 3
value = [0, 3, 0]
class = fake
impers ≤ 3.5
gini = 0.5
samples = 2
value = [1, 1, 0]
class = trusted
gini = 0.0
samples = 3
value = [0, 3, 0]
class = fake
gini = 0.0
samples = 2
value = [2, 0, 0]
class = trusted
gini = 0.0
samples = 1
value = [0, 0, 1]
class = parodic
gini = 0.0
samples = 10
value = [10, 0, 0]
class = trusted
gini = 0.0
samples = 1
value = [0, 0, 1]
class = parodic
ponctuation ≤ 24.5
gini = 0.667
samples = 9
value = [3, 3, 3]
class = trusted
On ≤ 0.5
gini = 0.42
samples = 20
value = [14, 6, 0]
class = trusted
adv ≤ 5.5
gini = 0.5
samples = 2
value = [0, 1, 1]
class = fake
gini = 0.0
samples = 2
value = [2, 0, 0]
class = trusted
gini = 0.0
samples = 1
value = [0, 1, 0]
class = fake
gini = 0.0
samples = 1
value = [0, 0, 1]
class = parodic
suspen ≤ 0.5
gini = 0.375
samples = 4
value = [3, 1, 0]
class = trusted
ponctuation ≤ 26.5
gini = 0.48
samples = 5
value = [0, 2, 3]
class = parodic
ponctuation ≤ 40.5
gini = 0.49
samples = 7
value = [3, 4, 0]
class = fake
presse ≤ 2.5
gini = 0.26
samples = 13
value = [11, 2, 0]
class = trusted
ponctuation ≤ 22.5
gini = 0.5
samples = 2
value = [1, 1, 0]
class = trusted
gini = 0.0
samples = 2
value = [2, 0, 0]
class = trusted
gini = 0.0
samples = 2
value = [0, 0, 2]
class = parodic
selon ≤ 1.0
gini = 0.444
samples = 3
value = [0, 2, 1]
class = fake
gini = 0.0
samples = 3
value = [3, 0, 0]
class = trusted
gini = 0.0
samples = 4
value = [0, 4, 0]
class = fake
adv ≤ 3.5
gini = 0.153
samples = 12
value = [11, 1, 0]
class = trusted
gini = 0.0
samples = 1
value = [0, 1, 0]
class = fake
gini = 0.0
samples = 1
value = [1, 0, 0]
class = trusted
gini = 0.0
samples = 1
value = [0, 1, 0]
class = fake
gini = 0.0
samples = 2
value = [0, 2, 0]
class = fake
gini = 0.0
samples = 1
value = [0, 0, 1]
class = parodic
ponctuation ≤ 57.0
gini = 0.375
samples = 4
value = [3, 1, 0]
class = trusted
gini = 0.0
samples = 8
value = [8, 0, 0]
class = trusted
gini = 0.0
samples = 3
value = [3, 0, 0]
class = trusted
gini = 0.0
samples = 1
value = [0, 1, 0]
class = fake