# Classification de tweets

BUGNON Florentin DUVIVIER Corentin

Décembre 2022

## Table des matières

# 1 Description générale du projet

L'objectif de ce projet a été de construire un programme pouvant récupérer depuis Twitter des tweets et d'en permettre l'analyse avec des algorithmes de machine learning afin d'en prédire une polarité, négatif, positif ou neutre.

L'application est faite de 2 modules principaux différent, backend et frontend, backend contient tout les algorithmes dont nous auront besoin, que ce soit pour la récupération des tweets, le nettoyage des données ou bien la classification. Le code y est écrit en python.

Le second module contient la vue, et les éléments avec lequel l'utilisateur peut interagir. Le code y est écrit en react-js.

Notre application à été conçu selon le modèle MVC (modèle vue contrôleur)

L'utilisateur interagit avec la vue, qui communiquera avec la partie logique de l'application via une API, qui répondra en envoyant les résultats des algorithmes utilisés.

Dépôt utilisé :

```
https://github.com/Xwooz/PJE_twitter.git
```

# 2 Récupération et nettoyage des données

## 2.1 Récupération des données

Les données requises pour notre classification sont ici les textes contenu dans les tweets. Pour les récupérer, il nous faut passer par l'API de twitter.

Il y a plusieurs interfaces pour récupérer ces tweets, nous avons choisi "Twitter API V2" en envoyant directement des requêtes HTTP. Ce choix nous permet une grande flexibilité dans nos requêtes, que peu de librairies (Comme twitter4j ou pytweet) nous offrait.

Les requêtes sont donc envoyées vers 'https://api.twitter.com/2/tweets/search/recent', en construisant la requête selon les mot clés que nous voulons, la langue des tweets configurés, le nombre maximal de tweets récupérés ou encore le # des tweets.

Un exemple de requête est : https://api.twitter.com/2/tweets/search/recent?query=%28Equipe%20de%20France%29%2
is%3Aretweet%20lang%3Afr&max_results=100&tweet.fields=created_at,id,text

Une fois les données récupérées, nous les avons stockées sous forme de fichiers csv.

## 2.2 Nettoyage des données

Maintenant que nous avons récupérer des tweets, le but est alors de les nettoyer afin d'en former une base de données uniforme qui pourra nourrir nos algorithmes de machine learning en le biaisant un minimum.

Pour ce faire, nous avons créer une chaîne de traitements utilisant le regex pour nettoyer ces tweets.

Nous avons utilisés 8 traitements différents :

- Remplacement des retweets avec de simple '@'
- Remplacement des username avec de simple '@'
- Remplacement des url par le mot 'url'
- Ajout d'espace avant et après chaque élément de ponctuation
- Remplacement des tabulations avec de simples espaces
- Remplacement des prix en dollars par sa valeur sans signe
- Remplacement des prix en euros par sa valeur sans signe
- Suppression des groupes d'espaces multiples avec un unique

Ces 8 traitements nous permette d'avoir une base données de base et de futurs extractions comparables en éliminant le plus d'éléments parasites pour nos algorithmes.

# 3 Classification

La classification est la partie centrale de ce projet. Pour rappel, le but est de classifier chaque tweets entre 3 classes : neutre, positif ou négatif.

Pour chaque algorithmes ci-dessous, on considère les macros NEUTRE, POSITIF et NEGATIF définies.

Il est important de préciser que les bases de données que nous avons construite pour chaque sujets n'ont pas été annoté à la main mais grâce au premier algorithme naïf que nous allons présenter ci-dessous. Cela implique donc que les résultats obtenus grâce aux algorithmes de KNN ou Bayes et variantes vont entièrement dépendre de cet algorithme naïf.

Pour comparer les résultats de nos algorithmes, nous avons construit des ensembles d'entraînement et de test. Tous sont de même taille, c'est à dire 2/3 1/3.

Les matrices de confusions et scores seront donc calculés sur la classification de notre ensemble de test par rapport à la classification naïve.

## 3.1 Mot clés

### 3.1.1 Algorithme

Le premier moyen de classification est une classification naïve par mot clés.

Nous avons, grâce à une liste de mots clés fournie, classifiés nos tweets dans la classe correspondante. Cette liste contient une série de mots considérés comme négatifs, et une autre série de mots considérés comme positifs.

En cherchant les occurrences de ces mots dans nos tweets, nous avons pu déterminer si ceux-ci sont positif ou négatif en regardant la proportion d'occurrences de mots négatif ou positif.

Si le nombre d'occurrences de mot connotés positif est supérieur à celui de mot connotés négatif, alors la classification du tweet sera 'positif'. Si c'est le nombre d'occurrences de mot connotés négatif qui est supérieur à celui de mot connotés positif, la classification de tweet sera alors 'négatif'. si ces proportions sont égales, le tweet sera donc classifié 'neutre'.

Voici l'algorithme :

```
annote(tweet) :
    # POSITIFS liste des mots positifs
    # NEGATIFS liste des mots négatifs
    p = 0
    n = 0
    pour chaque mot w dans POSITIFS :
        si le mot est dans tweet :
            p = p + 1
    pour chaque mot w dans NEGATIFS :
        si le mot est dans tweet :
            n = n + 1
    score = p - n
    si score > 0 :
        return POSITIF
    si score < 0 :
```

```
            return NEGATIF
        return NEUTRE
```

### 3.1.2 Résultats

Cette méthode étant très naïve, nous savions que cela ne nous donnerait pas de résultats optimaux.

Des mots dans la liste de mots positif comme ’je veux’, ’retour’ ou ’ok’ sont discutables et pourraient très bien être utilisé dans des phrases plutôt neutres.

D’autres mots de la liste de mots positifs comme ’idiot’, ’loufoque’ ou ’pitié’ sont eux plutôt négatifs et ne devraient pas faire partie de cette liste.

Nous avons cependant choisi de garder cette liste tel quelle et de faire fonctionner notre algorithme avec celle-ci.

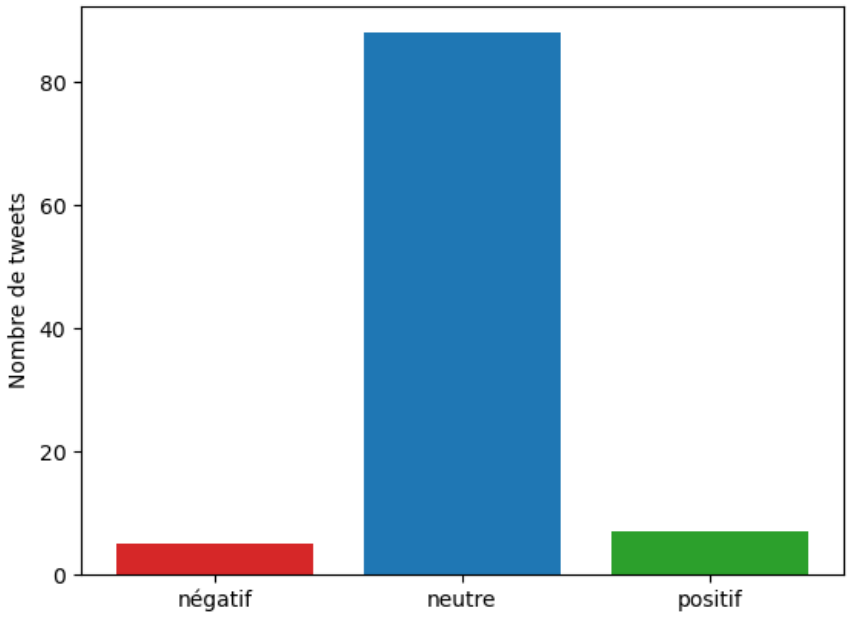


Figure 1: Nombre de tweets annotés dans les différentes classes par l’algorithme de classification naïf

Nous obtenons une classification des tweets majoritairement neutre.

## 3.2 KNN

La seconde méthode de classification utilisé est le KNN

### 3.2.1 Méthode et algorithme

La méthode du KNN (k-nearest-neighbors) consiste à prendre en compte les k plus proche voisins d'un exemple dans une base de données annotés, et d'annoter notre exemple selon la plus grande représentation de classe dans nos k plus proche voisins.

Dans notre cas, nous allons donc devoir implémenter une fonction de distance entre 2 tweets, et d'annoter un tweet en fonction de ses k-voisins les plus proches selon cette fonction de distance.

La fonction de distance implémentée est une fonction de distance naïve, elle correspond à : distance(m1,m2) = (Nombre total de mots – nombre de mots communs) / nombre total de mots

Nous devons également déterminer le k optimal pour notre algorithme. Les résultats des tests sont la partie résultats.

```
Données : x le tweet à étiqueter, k le nombre de voisins
pour i allant de 1 à k
mettre le point i dans proches_voisins
fin pour
pour i allant de k+1 à N
si la distance entre i et x est inférieure à la distance d'un des
points de proches_voisins à x
supprimer de proches_voisins le point le plus éloigné de x
mettre dans proches_voisins le point i
fin si
fin pour
proches_voisins contient les k plus proches voisins de x
vote(proche_voisins) donnent la classe majoritaire des voisin
```

### 3.2.2 Résultats

Voici les résultats de notre algorithme pour différents réglages du k.

| k = 1 | Polarités estimés | | |
|---|---|---|---|
| Polarité réelle | négatif | neutre | positif |
| Négatif | 0 | 2 | 0 |
| Neutre | 0 | 19 | 7 |
| Positif | 0 | 6 | 0 |

| k = 3 | Polarités estimés | | |
|---|---|---|---|
| Polarité réelle | négatif | neutre | positif |
| Négatif | 0 | 2 | 0 |
| Neutre | 0 | 24 | 1 |
| Positif | 0 | 6 | 0 |

| k = 8 | Polarités estimés | | |
|---|---|---|---|
| Polarité réelle | négatif | neutre | positif |
| Négatif | 0 | 2 | 0 |
| Neutre | 0 | 26 | 0 |
| Positif | 0 | 6 | 0 |

Une hypothèse que l'on peut faire sur l'algorithme du knn, et que plus le nombre de voisins est grand, plus la précision sera importante si la taille de l'ensemble est suffisante.

Or dans notre cas, la base de tweets est sur-représentée par des tweets classifiés neutres. Avoir un k plus important va donc augmenter le nombre de voisin classifiés neutre et donc biaiser la classe majoritaire des voisin, jusqu'à conduire a une classification des tweets en neutre à une probabilité de 1, comme on le voit pour k = 8.

Avec la validation croisée avec 10 folds, nous avons pu déterminer que le meilleur k est 2 pour un taux d'erreurs de 0,019.

## 3.3 BAYES et variantes

La dernière méthode utilisée est une méthode de classification probabiliste, la classification bayésienne.

### 3.3.1 Méthode et algorithme

La méthode de classification bayésienne consiste à calculer les probabilités que notre exemple soit d'une classe c pour chaque classes C (ici, neutre, positif et négatif).

L'algorithme calcul donc :

$$c(t) = \underset{c \to C}{\arg\max}\, P(c|t) \tag{1}$$

L'algorithme que nous avons implémenté pour classification bayésienne par présence et par unigramme est donc le suivant :

```
calcule_probabilités(tweet, base_apprentissage):
    // base commune aux différentes versions

    CLASSES = [NEUTRE, POSITIF, NEGATIF]

    count_negatif = taille(base_apprentissage[base_apprentissage['annote'] == NEGATIF])
    count_neutre = taille(base_apprentissage[base_apprentissage['annote'] == NEUTRE])
    count_positif = taille(base_apprentissage[base_apprentissage['annote'] == POSITIF])

    proba[NEGATIF] = count_negatif / taille(base_apprentissage)
    proba[NEUTRE] = count_neutre / taille(base_apprentissage)
    proba[POSITIF] = count_positif /  taille(base_apprentissage)

    // Calcul de présence
    // Pour les unigrammes
```

```
pour chaque classe dans CLASSES :
    compteur_occurence = 0
    compteur_mot_classe = 0
    pour chaque unigramme dans tweet :

        pour chaque tweet_base dans base_apprentissage[base_apprentissage['annote']
        == classe] :
            compteur_occurence += tweet_base.count(unigramme)
            compteur_mot_classe += taille(mots dans tweet_base)
    proba[classe] *= compteur_occurence / compteur_mot_classe
```

Des variantes ont été implémentées, comme la possibilité de calculer les probabilités grâce à la fréquence d'apparition des mots au lieu du nombre d'apparition, ou encore de mener ces calculs sur les bi-grammes (groupes de 2 mots) à la place des uni-grammes (ou utiliser les deux simultanément.

### 3.3.2 Résultats

Pour mesurer la qualité de ces deux classifieurs, nous avons utilisé la validation croisée, avec un nombre de fold à 10.

Les résultats sont les suivants :

| Représentation | Ensembles considérés | taux d'erreur |
|---|---|---|
| présence | uni-grammes | 0.0144 |
| présence | bi-grammes | 0.1066 |
| présence | uni-grammes et bi-grammes | 0.0266 |
| fréquence | uni-grammes | 0.0144 |
| fréquence | bi-grammes | 0.0144 |
| fréquence | uni-grammes et bi-grammes | 0.0144 |

Selon ces résultats, la méthode la plus efficace est la classification bayesienne par fréquence, quel que soit l'ensemble de mots considérés.

# 4 Application

## 4.1 Installation

```
git clone https://github.com/Xwooz/PJE_twitter.git
```

Démarrage de l'application:

```
PJE_twitter/backend $ uvicorn main:app --reload

puis dans un autre terminal :

PJE_twitter/front-end $ npm i
PJE_twitter/front-end $ npm run start
```

Une page web devrait s'ouvrir dans le navigateur

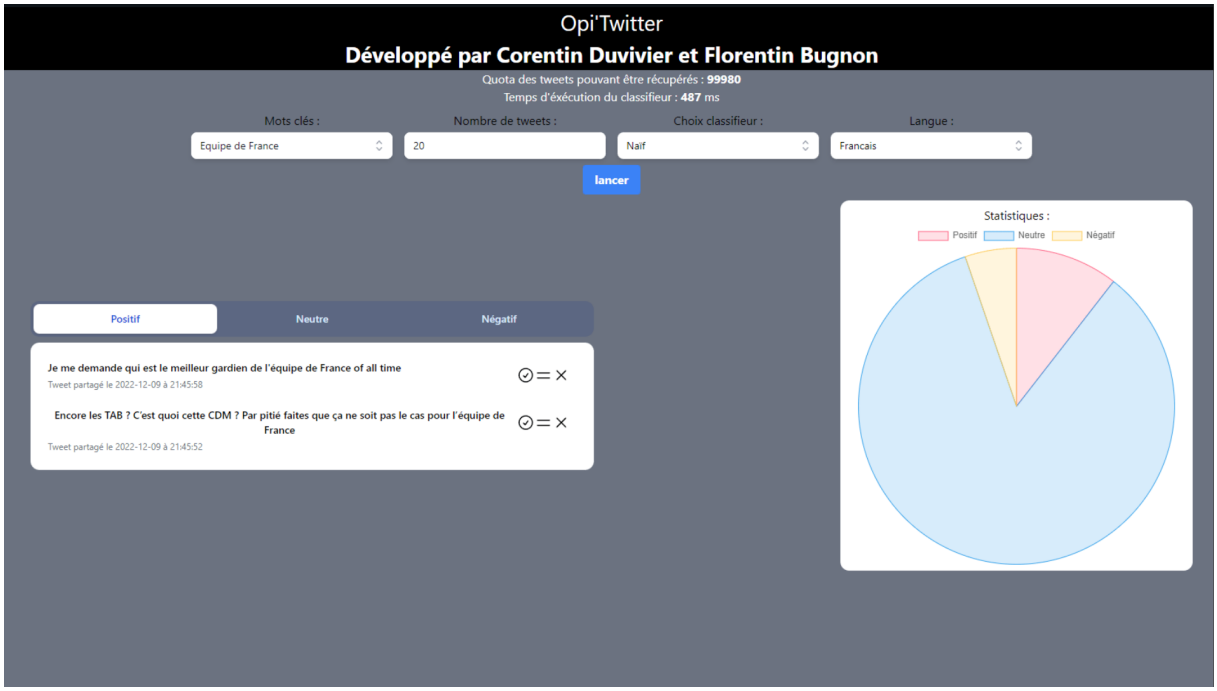


Figure 2: Capture d'écran de l'interface graphique de l'application

## 4.2 Fonctionnalités

Pour classifier des tweets, plusieurs actions de l'utilisateur sont à performer.

- Sélectionner un mot clé
- Sélectionner le nombre de tweets à récupérer
- Choisir le classifieur à utiliser
- Choisir une langue de recherche des tweets

Informations disponible sur la classification :

- Temps d'exécution du classifieur utilisé

- Statistiques de la répartition des classes
- Résultats de la classification

## 4.3 Utilisation

Après avoir sélectionné les paramètres et appuyé sur lancer, l'algorithme de classification correspondant sera lancé, avec la base de données associés au sujet choisi, pour classifier les tweets récupérés.

Les tweets ainsi classifiés seront affichés dans les trois classes correspondantes. Si l'utilisateur ne trouve pas la classification juste, il pourra ré-annoter à la main les tweets correspondant.

Sur la droite de l'écran, il y a également un graphique qui récapitule les statistiques de la classification.