PJE
#currentmood

Quentin Van de Kadsye Jérôme Tanghe

Mardi 15 décembre 2015

Table des matières

1 Le projet **2**

2 L'interface de #currentmood **4**

3 API Twitter et gestion des tweets **5**

3.1 La classe `CMTwitter` 5

3.2 La classe `Tweet` 7

4 La base d'apprentissage **8**

4.1 Le nettoyage des tweets 8

4.2 Le fichier CSV 8

4.3 Classifier manuellement un tweet dans #currentmood 9

5 Classification automatique des tweets **10**

5.1 Classification par mots-clés 10

5.2 Classification par la méthode KNN 11

5.3 Classification par la méthode bayésienne 12

6 Conclusion **15**

I – Le projet

Il est parfois intéressant pour les entreprises de connaître l'humeur générale des gens concernant un sujet donné. Twitter étant une plateforme où l'on peut s'exprimer librement, c'est donc un emplacement de choix pour récolter ce type d'information. Se pose alors le problème suivant : comment connaître rapidement l'humeur des personnes sur un sujet donné ?

Nommé selon le hashtag du même nom, **#currentmood** est un programme tentant de répondre à ce besoin. Écrit en Java, il permet d'estimer l'humeur d'un ou plusieurs messages publiés sur Twitter (*tweet*), à l'aide d'une des trois méthodes proposées :

— **Mots-clés :** utilise une liste de mots prédéfinis dans des fichiers pour déterminer l'humeur du tweet.
— **KNN :** évalue l'humeur d'un tweet en fonction de l'humeur de k autres tweets, en recherchant dans une base de données de tweets dont on connaît déjà l'humeur ceux qui contiennent les mêmes mots.
— **Classification bayésienne :** évalue la probabilité d'humeur d'un tweet en calculant la probabilité que les mots qu'il contient appartiennent à cette humeur à partir de la base de données.

Le code source du logiciel est disponible sur GitHub : github.com/Deuchnord/currentmood.

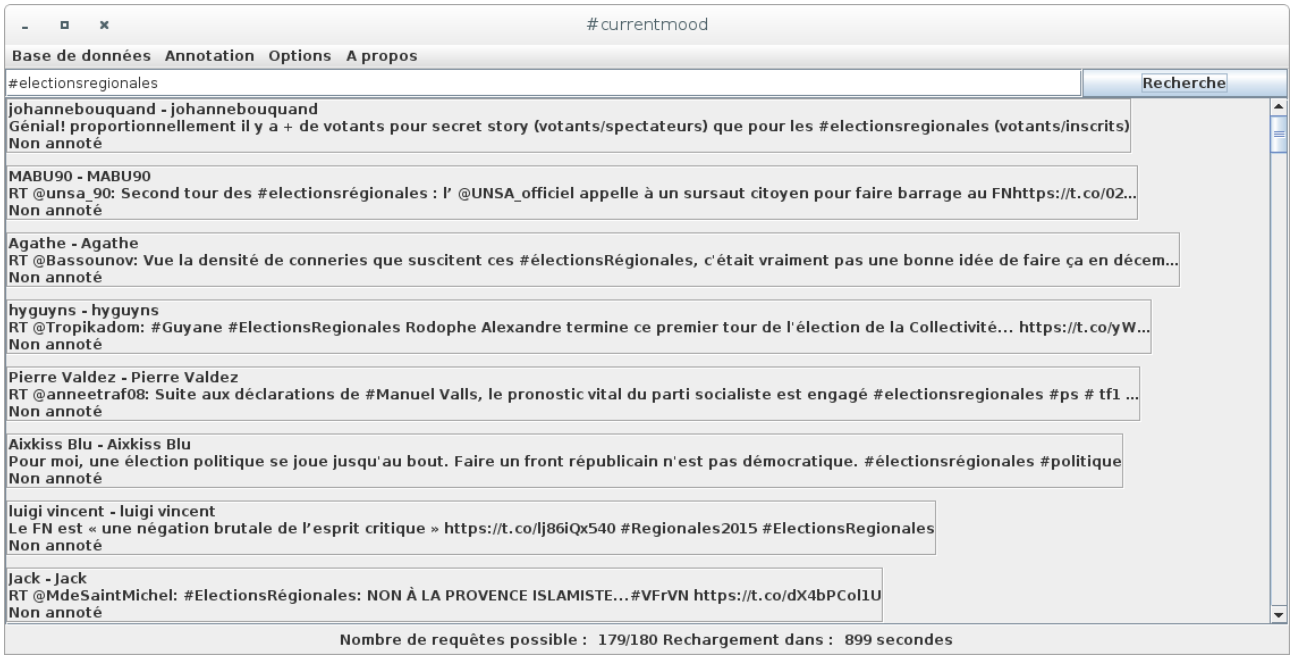

FIGURE 1.1 – Interface utilisateur générale de #currentmood

II – L'interface de #currentmood

III – API Twitter et gestion des tweets

1 – La classe `CMTwitter`

Afin de communiquer avec l'API de Twitter, nous avons utilisé la librairie **Twitter4J** [1] qui propose les fonctionnalités dont nous avons besoin pour mener à bien le projet. Pour faciliter son implémentation, nous avons également créé une classe, `CMTwitter`, s'interfaçant entre notre application et Twitter4J, comme le montre la figure 3.1 (page 6).

Cela nous permet d'utiliser l'API de Twitter à l'aide de trois méthodes principales seulement au lieu d'une dizaine :

— `setProxy()` afin de donner les paramètres proxy à Twitter4J ;
— `connect()` afin d'établir la connexion avec l'API de Twitter ;
— `searchTweets()` afin d'utiliser la fonction de recherche de l'API de Twitter.

La méthode `reset()`, quant à elle, est utilisée par les méthodes précédentes afin de permettre d'effectuer plusieurs requêtes. En effet, nous avons pu remarquer que l'objet `ConfigurationBuilder` périme après chaque requête, nous obligeant à le recréer avant d'effectuer une nouvelle requête.

L'utilisation de cette classe s'effectue en trois étapes principales.

Tout d'abord, on configure si nécessaire les paramètres proxy à l'aide de la méthode `setProxy()`. Cette dernière donne lesdits paramètres à l'objet `ConfigurationBuilder`.

Ensuite, on appelle la méthode `connect()` qui utilise l'objet `ConfigurationBuilder` pour obtenir une instance de `Twitter`.

C'est cette instance qui est ensuite utilisée par `searchTweets()` qui instancie un objet de Twitter4J permettant d'obtenir des tweets correspondant à la recherche. Cette méthode retourne une liste d'objets `Status` provenant de la librairie Twitter4J.

1. twitter4j.org

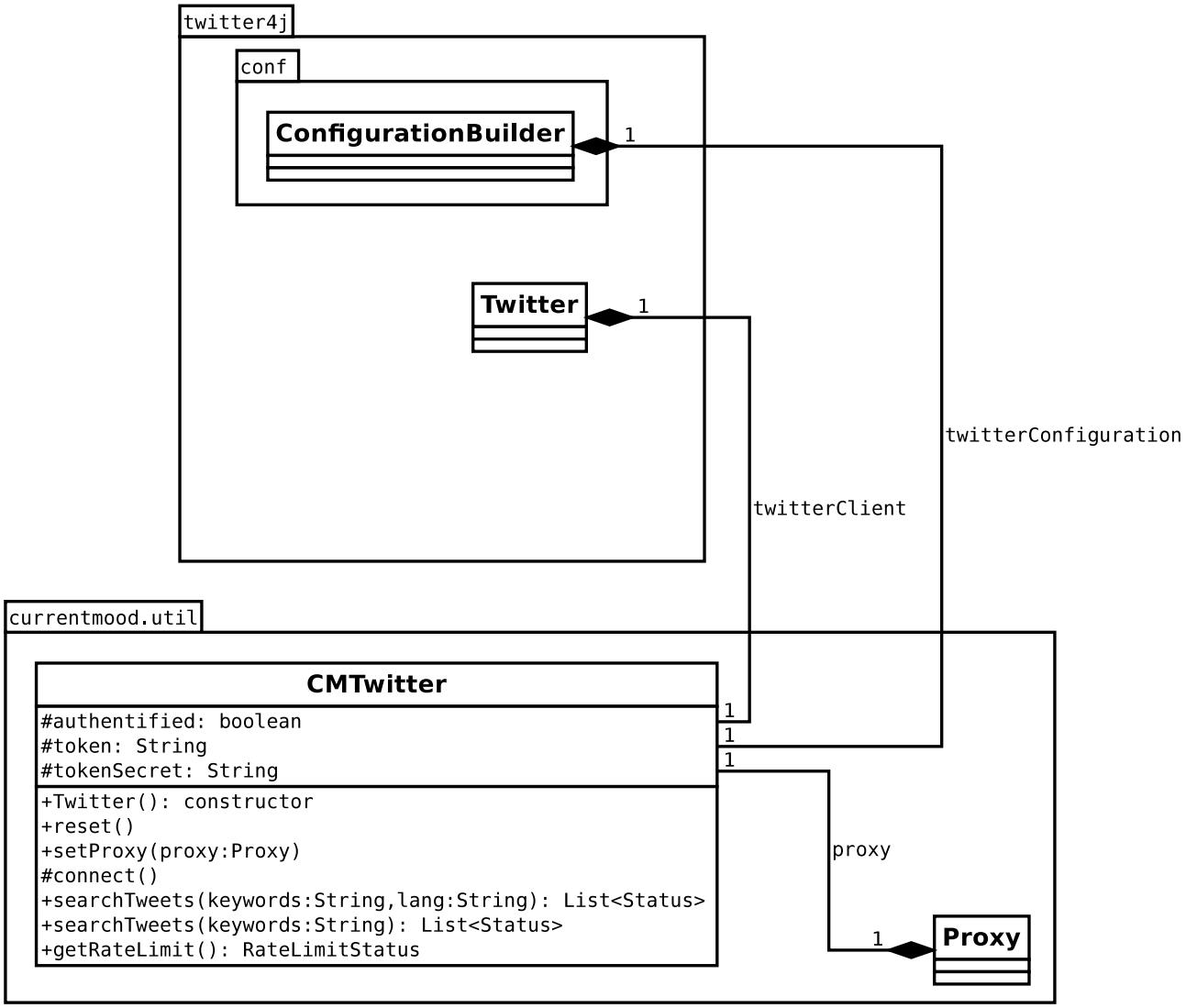

FIGURE 3.1 – Diagramme de classe montrant comment nous interfaçons Twitter4J

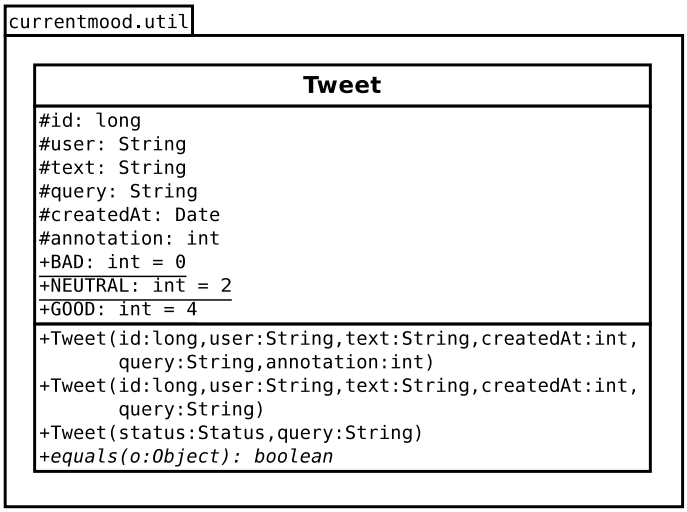

FIGURE 3.2 – La classe `Tweet`

2 – La classe `Tweet`

Nous nous sommes rapidement rendus compte que Twitter4J ne proposait aucune classe implémentant l'interface `Status` que nous devons manipuler. De plus, les objets que nous en obtenons contient de nombreuses informations qui ne nous sont pas utiles, tandis que d'autres nous étaient nécessaires mais n'y étaient pas présents.

Nous avons donc créé une classe indépendante de Twitter4J, `Tweet`, qui réponde à ce besoin (figure 3.2). Elle est utilisée dans toute l'application afin de contenir chaque tweet.

Cette classe est surtout composée d'accesseurs permettant d'accéder à chaque propriété du tweet. Ses deux premiers constructeurs permettent de créer un objet à partir de données déjà connues (typiquement lors de l'ouverture d'un fichier CSV), tandis que le troisième permet d'obtenir un objet `Tweet` à partir d'un objet `Status` généré par Twitter4J.

Nous avons également surchargé la méthode `equals()` de la super-classe `Object` afin de permettre la comparaison de l'objet courant avec un autre objet `Tweet`. Cela nous sera nécessaire pour certaines actions par la suite.

IV – La base d'apprentissage

1 – Le nettoyage des tweets

L'annotation automatique se basant sur une base d'apprentissage construite manuellement, il est primordial d'avoir une base de données saine pour éviter les erreurs d'interprétation. Cela passe par un nettoyage des tweets afin de limiter le « bruit ».

Lors de la sauvegarde des tweets annotés manuellement dans un fichier CSV, nous passons par une étape consistant à supprimer tout ce qui peut gêner l'interprétation du tweet ou la lecture du fichier CSV.

Ainsi, nous supprimons :

— Les retours à la ligne (pour respecter le format CSV)
— Les smileys `:)`, `:(`, `:D`, `:')`, `:'(` et `:'D`
— Les liens hypertextes
— Les `@usernames`
— Les hashtags
— La ponctuation
— Les symboles monnétaires principaux (€, $, £)
— Les pourcentages

Une fois ce nettoyage effectué, nous pouvons alors utiliser cette base de tweets pour annoter automatiquement d'autres tweets selon quatre méthodes qui seront traitées dans la partie 5 (page 10). Ce nettoyage permet également d'enregistrer les tweets dans un fichier CSV sans casser son format (en supprimant notamment les retours à la ligne et les virgules).

2 – Le fichier CSV

Pour gérer la base de données, il a été décidé de sauvegarder les messages annotés dans un fichier CSV [1]. Ce type de fichier a pour principal avantage d'être relativement léger et facile à lire par programmation.

1. *Comma-separated values*

Les données à sauvegarder étant globalement celles contenues dans les objets `Tweet`, l'ordre de sauvegarde sera donc le suivant :

1. Le numéro d'identification du tweet
2. Le nom de l'auteur du tweet
3. Le contenu du tweet
4. La date et l'heure du tweet, sous la forme d'un *timestamp*[2]. S'il est nul, il est ignoré.
5. La recherche qui a permis de trouver le tweet
6. L'annotation du tweet

L'annotation du tweet est enregistré sous la forme d'un nombre dont la valeur est précisé par le tableau 4.1.

Valeur de l'annotation	Humeur du tweet
0	Mauvais
2	Neutre
4	Bon

TABLE 4.1 – Valeur de l'annotation selon l'humeur du tweet

3 – Classifier manuellement un tweet dans #currentmood

Après avoir chargé des tweets provenant de Twitter dans #currentmood, il suffit de cliquer droit sur un tweet et, dans le menu contextuel, de choisir *Annoter à la main* et l'annotation que l'on souhaite ajouter au tweet (figure 4.1).

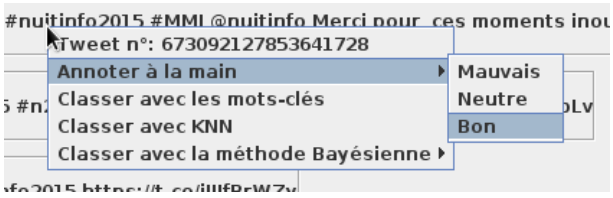

FIGURE 4.1 – Menu contextuel permettant d'annoter un tweet à la main

2. Nombre de secondes depuis le 1er janvier 1970, 0 h 00 min 00 s GMT

V – Classification automatique des tweets

1 – Classification par mots-clés

La classification par mots-clés est la plus simple des méthodes de classification. Elle consiste à se baser sur une liste de mots définis dans un fichier pour déterminer l'humeur d'un tweet.

Sa stratégie est des plus simples : pour un tweet donné, on recherche chacun de ses mots dans un dictionnaire définissant des mots positifs et des mots négatifs. Leur nombre respectifs déterminera alors l'humeur du tweet. Par exemple, un tweet contenant 3 mots considérés positifs et 5 mots considérés négatifs sera annoté négatif. Un tweet qui contiendrait un même nombre de mots positifs et négatifs est considéré comme étant neutre.

Résultats de la classification

Cet algorithme s'est révélé peu efficace du fait qu'il se contente de réduire une phrase complète à de simples mots auquel il donne un sens *a priori*. Il ne prend de ce fait pas compte du sens général du tweet. Prenons par exemple le tweet suivant :

Génial, ma banque m'a refusé mon crédit...

On remarque bien que ce tweet est plutôt négatif. Cependant, la présence du mot *génial* peut influencer la classification finale, ce qui peut déboucher, ici, à un faux positif.

Cet algorithme nécessite également que le dictionnaire utilisé contienne suffisamment de mots : en effet, un dictionnaire pas assez fourni signifierait que de nombreux tweets seraient soit mal annotés (en raison de la présence d'autres mots connus) ou annotés neutre.

Il est à ce titre intéressant de remarquer que le terme *neutre* n'a ici pas le même sens que dans les autres classifications. En effet, ici, *neutre* signifie que le tweet n'a pas pu être annoté soit parce qu'il ne contenait aucun mot connu du dictionnaire, soit parce qu'il contenait autant de mots positifs que de mots négatifs.qu'il contenait autant de mots positifs que de mots négatifs.

2 – Classification par la méthode KNN

La méthode des k plus proches voisins, ou KNN [1], est une méthode consistant à exploiter la base d'apprentissage en comptant le nombre de mots en commun entre un tweet à annoter et chaque tweet de la base.

L'humeur du tweet est alors déterminée selon l'humeur des k tweets avec lesquels il a le plus de mots en commun.

Pour ce faire, l'algorithme calcule la distance entre le tweet à annoter (que l'on nommera A par la suite) et chaque tweet de la base de données en conservant les k tweets les plus proches (k étant un nombre choisi par l'utilisateur). Il en déduit alors l'humeur du tweet A en comparant simplement le nombre de tweets positifs, neutres et négatifs parmi les k plus proches.

L'intérêt d'utiliser un nombre k est que l'on utilise alors un nombre réduit de tweets parmi ceux les plus proches, ce qui permet d'influer sur la précision de l'algorithme.

Cette méthode a pour principal avantage d'être simple à mettre en œuvre. Cependant, il peut être délicat de choisir une valeur pour k, qui peut grandement influer sur le résultat. De plus, comme pour la méthode de classification par mots-clés, le sens de la phrase n'est pas pris en compte. Pis, il peut aussi être perturbé par certains mots n'apportant pas de sens supplémentaire au message phrase et propres à la langue (articles, conjonctions de coordination, prépositions...).

Observations

De part sa conception, cette méthode est plutôt sensible à la répartition des tweets enregistrés dans la base de données. En effet, si une des humeurs y est majoritaire par rapport aux autres, elle influencera grandement le résultat.

Ainsi, nous avons pu constater qu'avec une base de données contenant environ 80% de tweets positifs, quasimment tous les tweets annotés par la méthode KNN seront considérés positifs.

Pour que la méthode soit efficace, nous avons fait des tests avec plusieurs valeurs de k, et nous avons pu remarquer que, pour une base de n tweets :

— si $k = 1$, la méthode est peu efficace, car elle n'utilise que *le* plus proche voisin. L'humeur de ce voisin est donc toujours le résultat, qui peut être faux ;
— si $k = n$, on utilise toute la base de données et cela revient presque à utiliser la méthode par mots-clés, puisque l'on prend la majorité de tous les tweets présents dans la base ;
— si $1 < k < n$, on utilisera un nombre plus ou moins conséquents de voisins, ce qui permet d'avoir une plus grande variété pour déterminer l'humeur du tweet.

1. k-nearest neighbor

3 – Classification par la méthode bayésienne

La méthode de classification bayésienne est une méthode de classification probabiliste. Dans le cas de notre application, elle servira à déterminer la probabilité qu'un tweet appartienne à l'une des classes établies précédemment, c'est-à-dire : mauvais, neutre ou bon.

Afin de pouvoir évaluer un tweet, il faut d'abord calculer P(c) : la probabilité d'appartenance à une classe (P(M), P(N), P(B)). Un tweet étant un ensemble de mots, il a fallu déterminer l'appartenance d'un mot à une classe. Pour cela il a été nécessaire de répertorier la classe de chaque mot de chaque tweet se trouvant dans la base de données pour chaque classe.

Ensuite avec une formule mathématique, on calcule la probabilité que le tweet appartienne à la classe mauvais, neutre ou bon.

— La méthode par présence ou fréquence en se servant de tous les mots et par unigramme.
— La méthode par présence ou fréquence en servant de tous les mots et par bigramme.
— La méthode par présence ou fréquence en enlevant tous les mots de moins de trois lettres (à, de...) et par unigramme.
— La méthode par présence ou fréquence en enlevant tous les mots de moins de trois lettres (à, de...) et par bigramme.

Que sont les unigrammes et les bigrammes ?

Les unigrammes et les bigrammes sont en réalité des ensembles de mots, un unigramme est donc un ensemble composé d'un seul et unique mot. Un bigramme est un ensemble composé de deux mots, Ainsi « Chef-d'œuvre » peut être considéré comme un bigramme.

Le fait d'enlever les mots de moins de trois lettres, comme les prépositions, permet d'avoir des probabilités plus fines : En effet, les probabilités étant calculées en fonction de la probabilité d'appartenance à une classe pour chaque mot, et qu'il est difficile d'attribuer une classe à un mot comme une préposition, le résultat peut être faussé.

Concernant la variante « fréquence » de la classification bayésienne, la différence réside dans le fait que la probabilité qu'un mot appartienne à une classe particulière est multiplié par le nombre de fois où le mot apparaît dans le tweet.

Observations

La méthode de classification bayésienne est en général une bonne méthode d'annotation et elle ne dépend que d'un seul paramètre : la base de données de Tweet (dans notre cas), contrairement à KNN qui dépend de k et de la base de données ou de la méthode par mot-clés qui ne dépend pas de la base, mais de deux listes de mots arbitraires.

Cependant, au cours de nos tests, nous avons pu constater quelques faiblesses pour cette méthode.

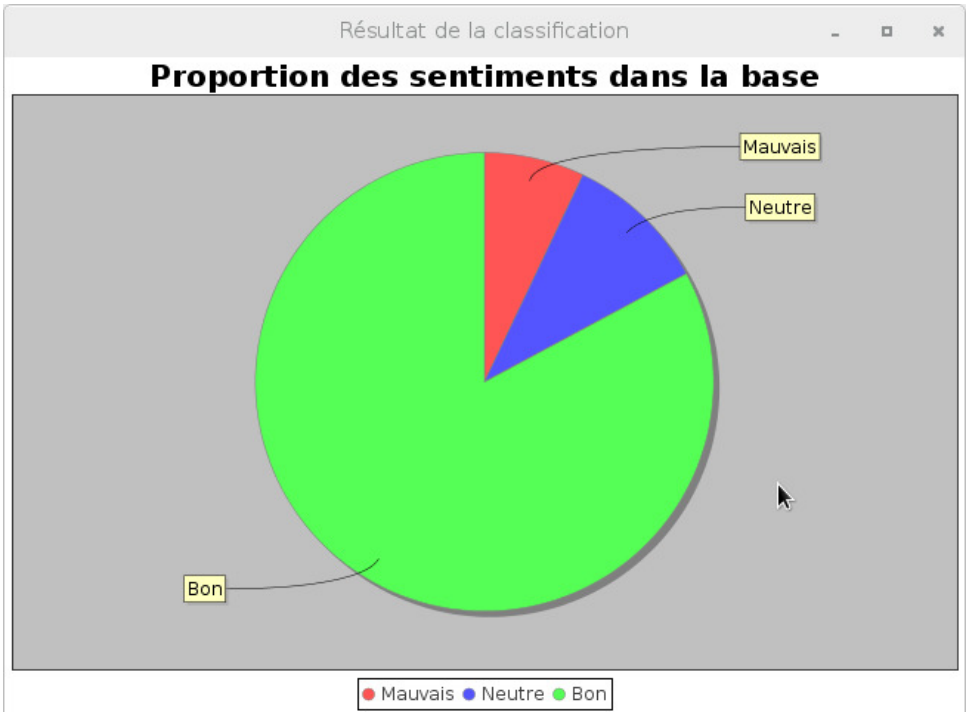

FIGURE 5.1 – Répartition des tweets dans une de nos bases de données

Comme on peut le voir sur la figure 5.1, cette base n'est pas équilibrée, puisqu'elle comporte trop de tweets « bons » par rapport aux autres classes.

Ainsi, pour le mot-clé « mort », on obtient les résultats représentés par la figure 5.2.

On pourrait très bien penser que le mot « mort » n'a pas forcément une connotation négative, prenons l'expression « Mort de rire ». Toutefois, le fait que algorithme retourne 100% de bon est clairement un problème.

Cela s'explique par le fait que la proportion de tweets Bon dans la base est trop grande, ainsi la probabilité qu'un mot soit « Bon » est affecté proportionnellement.

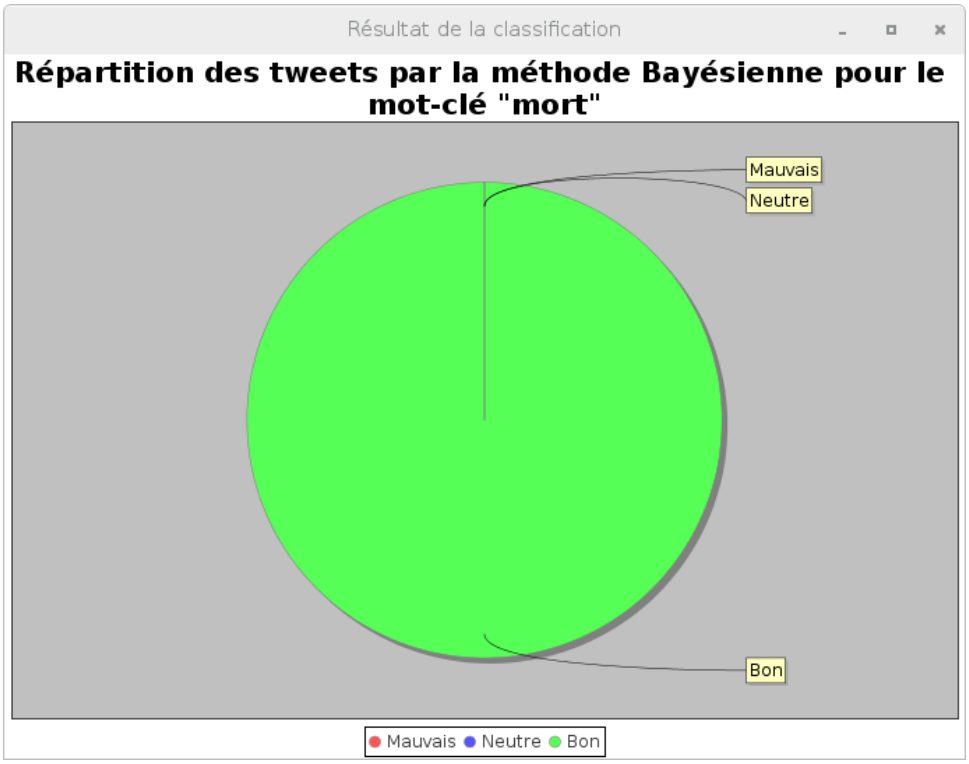

FIGURE 5.2 – Annotation automatique pour le mot-clé « mort »

VI – Conclusion