PJE
#currentmood

Quentin Van de Kadsye Jérôme Tanghe

Mardi 15 décembre 2015

Table des matières

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.

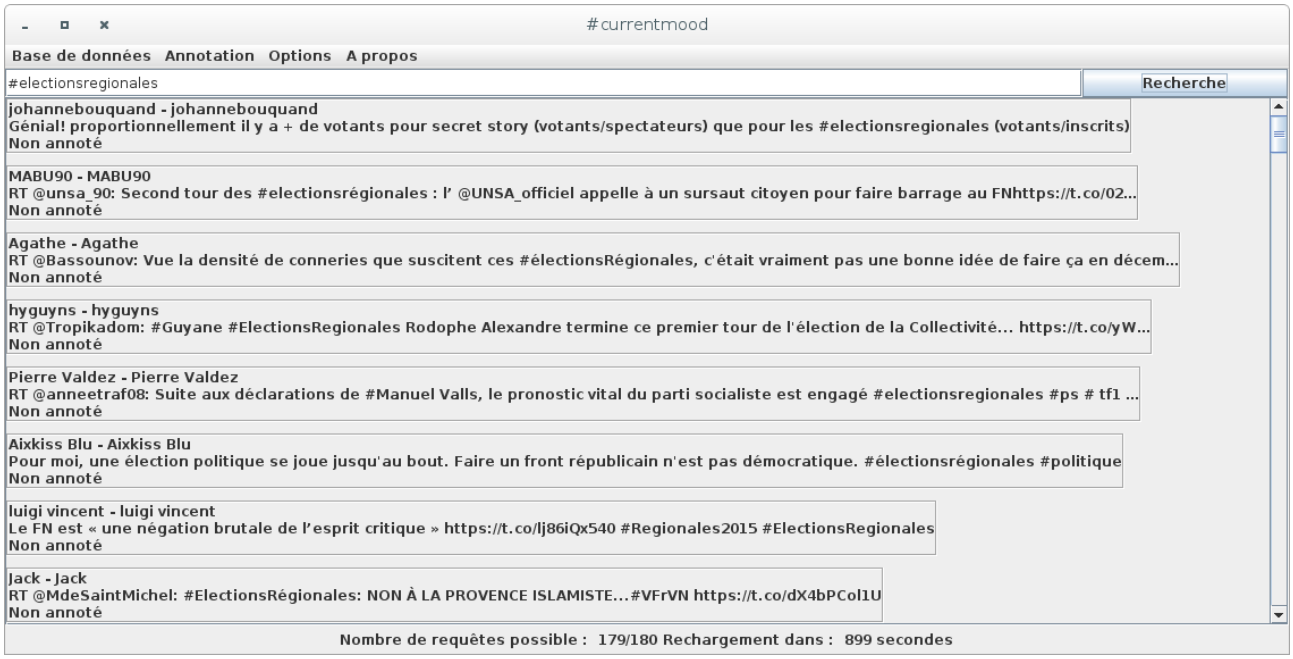

FIGURE 1.1 – Interface utilisateur générale de #currentmood

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

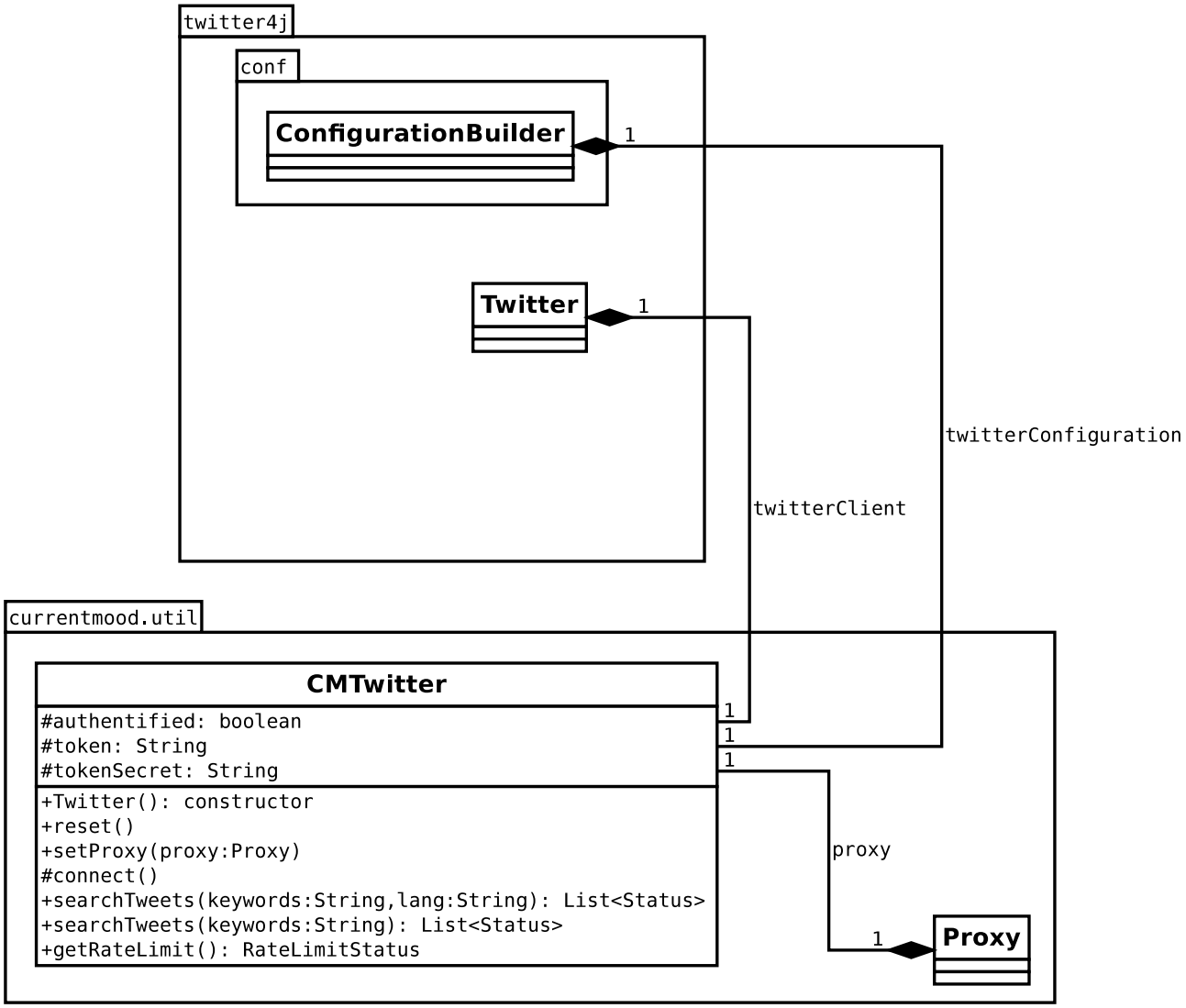

FIGURE 3.1 – Diagramme de classe montrant comment nous interfaçons Twitter4J

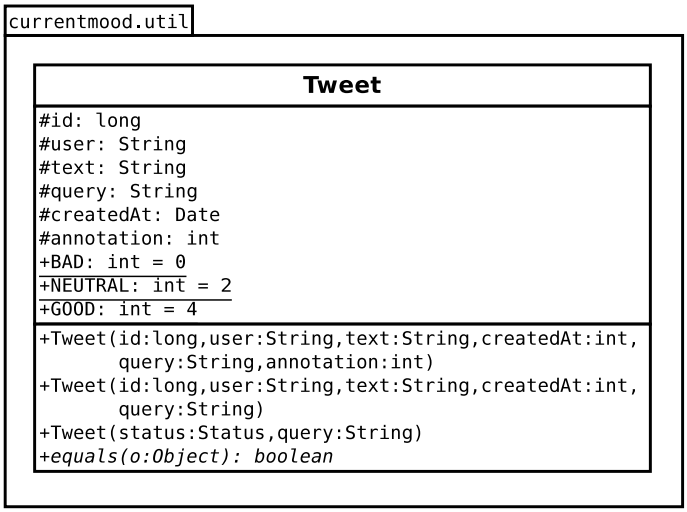

FIGURE 3.2 – La classe `Tweet`

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*

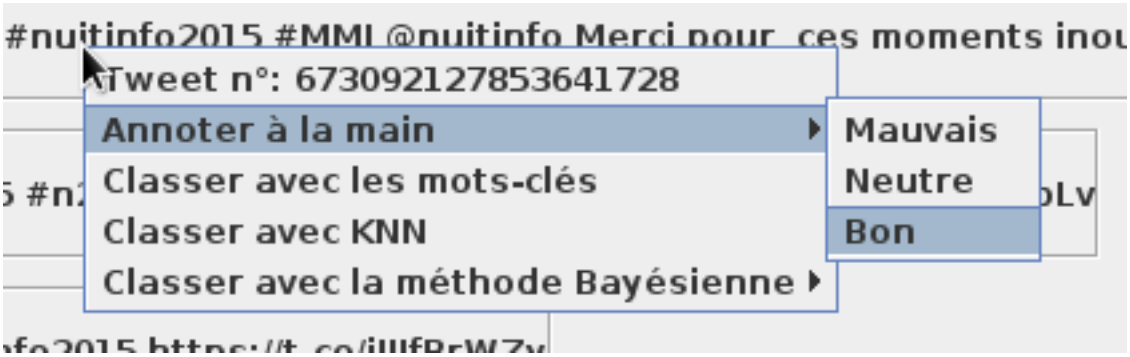

FIGURE 4.1 – Menu contextuel permettant d'annoter un tweet à la main

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).

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.

Cette méthode a pour principaux avantages d'être très simples à mettre en œuvre et de ne pas nécessiter de base d'apprentissage de tweets. Seul un dictionnaire de mots positifs et de mots négatifs est nécessaire pour pouvoir annoter un tweet.

Son inconvénient majeur est que le dictionnaire doit être particulièrement fourni en mots positifs et négatifs pour être efficace. De plus, réduire une phrase complète à de simples mots-clés ne permet pas de prendre en compte le sens général de la carte. 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.

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.

1. k-nearest neighbor

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...).

3 – Classification par la méthode bayésienne