Master 2 Informatique
TER Semestre 2

Prévention des suicides via Twitter

Encadrants :

Sandra BRINGAY
Jérôme AZÉ
Pascal PONCELET

Réalisé par :

Amayas ABBOUTE
Yasser BOUDJERIOU
Gilles ENTRINGER

REMERCIEMENTS

Nous tenons à remercier nos encadrants Sandra Bringay, Jérôme Azé et Pascal Poncelet de nous avoir guidés et accompagnés tout au long de notre travail.
Merci aussi à Amine Abdaoui et Julien Rabatel pour leurs conseils et leur aide.

SOMMAIRE

PARTIE 1 INTRODUCTION

Dans le cadre du TER du Master 2 Informatique à l'Université Montpellier 2, nous avons développé une application permettant d'utiliser le réseau social Twitter pour la prévention des suicides en ligne via ce dernier. Une interface de présentation permet de visualiser les résultats de notre travail.

Nous avons été encadrés par trois chercheurs du LIRMM (*Laboratoire Informatique de Robotique et de Micro-électronique de Montpellier*), à savoir, Sandra Bringay, Jérome Azé et Pascal Poncelet.

Le rapport s'organise comme suit :
Nous commençons par présenter le contexte général, les différents besoins ainsi que les objectifs du projet (Partie 2).

Dans un second temps, dans la partie 3, nous présentons le vocabulaire associé aux thématiques critiques. Nous présentons également les différents traitements qui sont effectués par l'application ainsi qu'une maquette du viewer que nous avons développé. Enfin, nous présentons le schéma de la base de données que nous avons mis en place ainsi que l'architecture de l'application.

Par la suite, nous expliquons successivement les outils et API que nous avons utilisés afin de mener à bien notre projet (Partie 4).

Dans la partie 5, nous présentons les différentes étapes liées à la réalisation de notre application. La partie 6, quant à elle, concerne la gestion et la planification du projet.

Finalement, la partie 7 est divisée en trois sections : les perspectives de notre projet, les difficultés que nous avons rencontrées ainsi qu'une conclusion générale.

PARTIE 2 PRÉSENTATION GÉNÉRALE

2.1 Contexte du projet

En France, chaque année, près de 10 500 personnes meurent par suicide. De plus, environ 220 000 tentatives de suicides sont prises en charge chaque année par les urgences hospitalières [1]. Le fardeau économique du suicide est estimé à 5 milliards d'euros pour l'année 2009 en France. Le suicide est donc un problème de santé publique majeur aux fortes conséquences socio-économiques. Aux États-Unis, le "Center for Disease Control and Prevention" [2] a compté plus de 38 000 suicides pour l'année 2010.

Dans ce contexte, les médias sociaux comme Twitter ou Facebook sont de plus en plus associés à des phénomènes tel que le harcèlement, l'intimidation ou même le suicide. Il est donc très important de détecter les potentielles victimes au plus tôt afin de pouvoir renforcer la prévention du suicide sur le web. En effet, nous pouvons citer en guise d'exemple les cas des deux rappeurs américains Freddy E. [3](âgé de 22 ans) et Capital Steez [4] (âgé de 19 ans) qui se sont donnés la mort en commentant en direct leurs actes sur leurs comptes Twitter.

Des études récentes de l'Université Brigham Young [5] ont démontré que Twitter pourrait être utilisé pour prévenir le suicide en ligne. Les thèmes évoqués et les termes utilisés par les dépressifs et suicidaires sont bien connus. Par exemple, ces personnes sont souvent victimes de harcèlement ou de cyber-intimidation. Ce fut le cas pour Hannah Smith, qui s'est pendue après avoir subie des abus sur le site Ask.fm [6]. Twitter permettrait ainsi, de mettre en place une surveillance en temps réel par rapport aux différents facteurs de risque. De plus, cette étude a démontré que le nombre de tweets suicidaires était fortement corrélé avec le taux de suicide réel.

2.2 Besoins

Les besoins pour ce projet peuvent être divisés en quatre parties : les besoins liées à la construction d'un vocabulaire associé à la thématique du suicide, la récupération des messages (appelés "tweets" par la suite) à partir de Twitter, les besoins liées à l'outil de classification automatique et les besoins relatifs à une interface permettant de présenter nos résultats.

1. http ://www.sante.gouv.fr/etat-des-lieux-du-suicide-en-france.html
2. http ://www.cdc.gov/nchs/fastats/suicide.htm
3. https ://twitter.com/Freddy_E
4. https ://twitter.com/CapitalSTEEZ_
5. http ://news.byu.edu/archive13-oct-suicide.aspx
6. http ://ask.fm/

2.2.1 Vocabulaire

Avant d'entamer la partie développement du projet, nous étions amenés à définir un vocabulaire associé aux différentes thématiques critiques liées au suicide (e.g. dépression, peur, harcèlement, etc.). Du fait que la plupart des messages soient publiés en anglais sur Twitter, il était préférable de définir le vocabulaire dans cette même langue.
De plus, le vocabulaire doit être divisé en différentes catégories et sous-catégories afin de pouvoir identifier facilement le degré de menace provenant du tweet. Par exemple, dans le cas d'un harcèlement via un tweet, il est nécessaire d'identifier les destinataires des tweets (car ce sont eux qui risquent de passer à l'acte). Au contraire, pour les autres catégories du vocabulaire, ce sont les personnes ayant publiées le tweet qui doivent être identifiées.

Pour stocker le vocabulaire, l'application doit comporter une base de données, qui sera également utilisée pour stocker les tweets suspects et les résultats de la classification.

2.2.2 Récupération des tweets

La collecte des tweets est un deuxième besoin de notre projet. L'application à développer doit permettre d'identifier automatiquement les tweets suspects (à partir du vocabulaire défini auparavant) et de les stocker pour une analyse approfondie dans la base de données de l'application.

2.2.3 Outil de classification automatique

L'intégration d'un outil de classification automatique dans l'application permet d'effectuer une analyse approfondie et automatisée des "tweets suspects" stockés dans la base de données. Nous avons identifié deux classes :

1. Les tweets suspects pour lesquels il y a un risque élevé que les auteurs passent à l'acte.
2. Les tweets suspects sans risque c'est-à-dire sans que pour autant l'auteur du tweet ne passe à l'acte.

2.2.4 Interface de présentation

Ce projet peut intéresser un grand nombre d'experts issus de différents domaines (psychologues, médecins, etc.). Il est nécessaire pour ces derniers de développer un viewer permettant de représenter les différents résultats du projet (les tweets suspects, le vocabulaire et des statistiques).

Une application Web est mieux adaptée a ce type de projet pour qu'elle soit accessible à un grand nombre de spécialistes.

2.3 Objectifs

À partir d'un vocabulaire associé à la thématique du suicide et des messages récupérés à partir de Twitter, l'application à développer doit permettre de classifier automatiquement les tweets et de lever des alertes.

En raison de la courte durée de ce TER, nous avons été amenés à développer un prototype pour une application qui va certainement évoluer dans le futur et qui peut faire partie d'un projet plus ambitieux. En résumé, notre principal rôle dans ce projet peut être décliné en cinq points essentiels :

— Conceptualisation et implémentation d'une base de données ;

— Définition et stockage dans la base de données d'un vocabulaire lié à la problématique du suicide ;

— Collecte et stockage des tweets suspects (comportant un élément du vocabulaire défini auparavant) ;

— Mise en place d'un outil de classification automatique et évaluation du classifieur sur les tweets suspects collectés auparavant ;

— Développement d'une interface web permettant de visualiser les résultats.

PARTIE 3 ÉTUDE ET CONCEPTION

3.1 Étude documentaire

À partir d'une étude documentaire, nous avons défini différentes thématiques (catégories) qui sont liées au suicide. Pour chaque thématique, nous avons collecté les termes associés que les utilisateurs de Twitter pourraient utiliser pour exprimer leur intention de se suicider.

Nous avons défini 7 thématiques comme étant fortement liées au suicide :

1. Dépression
2. Peur
3. Tristesse / Blessure psychologique
4. Solitude
5. Description du passage à l'acte
6. Insultes / Cyber-intimidation
7. Anorexie

De plus, nous avons collecté des phrases entières qui sont souvent utilisées par des gens qui ont l'intention de se suicider comme par exemple, "I want to die" ou "My family would be better off without me".

Voici la liste des sources qui nous ont permis de répertorier le vocabulaire :

— http://www.sba.pdx.edu/faculty/mblake/448/FeelingsList.pdf

— http://www.myvocabulary.com/word-list/depression-vocabulary/

— http://quizlet.com/7441476/mental-health-vocabulary-flash-cards/

— http://karlamclaren.com/wp-content/uploads/2010/11/Emotional-Vocabulary1.pdf

— http://www.slangsearch.com/insults.html

— http://www.myvocabulary.com/word-list/bullying-and-gangsvocabulary/

3.2 Modèle conceptuel des traitements

En nous basant sur les besoins et objectifs du projet, nous avons défini les traitements nécessaires pour le chargement du vocabulaire dans la base de données, pour le chargement des tweets et pour la classification des tweets. Les trois modèles conceptuels (cf. figures 3.1, 3.2, 3.3) suivants permettent de représenter l'ordre dans lequel sont exécutés les différents traitements.

3.2.1 Insertion du vocabulaire dans la base de données

Les différents termes qui composent le vocabulaire ont été stockés dans des fichiers CSV. Afin de stocker ces termes dans la base de données du projet, un parseur permet de charger les données sous Java. Par la suite, Hibernate et Java permettent de stocker les différents termes dans la base de données, à savoir, la table nommée "Thematics".

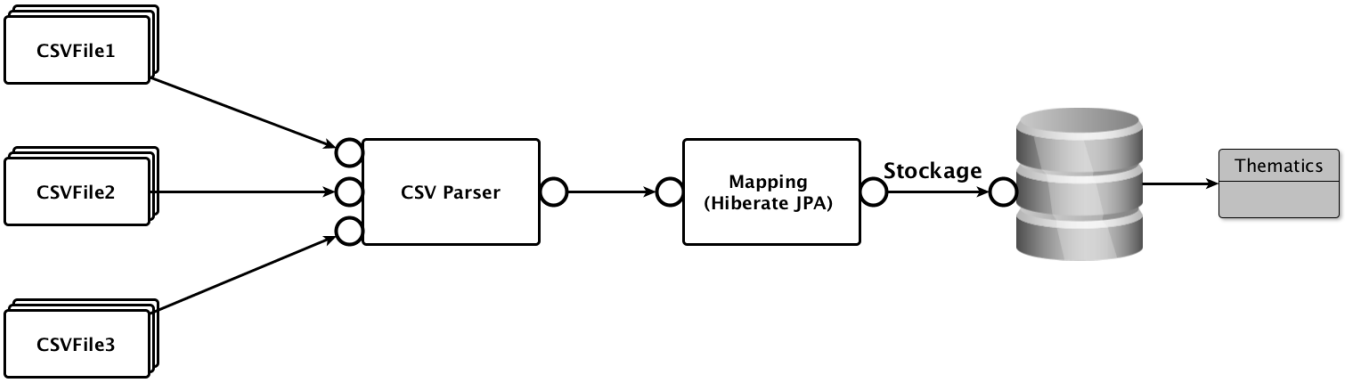

FIGURE 3.1 – Traitements pour stocker le vocabulaire dans la base de données

3.2.2 Chargement des Tweets

Afin de récupérer les tweets à partir de l'API Rest de Twitter, les traitements suivants doivent être effectués :

— Chargement du vocabulaire à partir de la base de données (utilisation de multi-threading afin d'accélérer le chargement)

— Recherche des tweets comportants au moins un terme ou une expression répertorié dans le vocabulaire utilisé

— Stockage des tweets suspects dans la base de données et mise à jour de la table tweets_thematics (table d'association) (Hibernate Mapping).

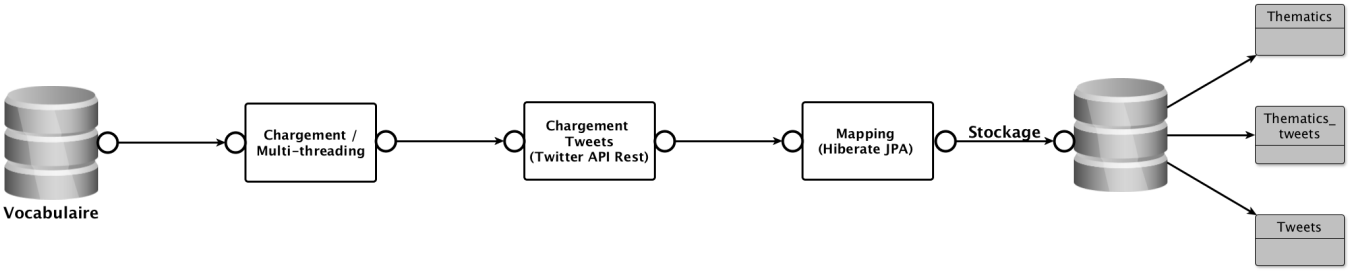

FIGURE 3.2 – Récupération et stockage des tweets suspects

3.2.3 Classification

Différents traitements permettent de classifier les tweets en 2 classes : Les tweets suspects à risque et les tweets suspects sans risques (cf. 2.2.3). Dans ce contexte, nous avons dans un premier temps créé un listing de 150 tweets correspondants à ces deux classes. Nous avons pris le soin de ne garder que des tweets pour lesquels nous pouvons affirmer avec certitude s'il y a un risque que l'auteur commette un suicide ou pas. Afin d'augmenter cette certitude, nous avons également ajouté des tweets de personnes ayant commis un suicide en le commentant sur le réseau social en question quelques minutes ou quelques heures avant (cf. Contexte du Projet 2.1).

Ces tweets "classifiés manuellement" ont été chargés dans l'application et Weka les utilise pour apprendre un modèle prédictif. Par la suite, Weka sera capable de classifier les tweets suspects (stockés dans la table "tweets") selon les deux classes. En sortie, l'application fournit donc les tweets suspects classifiés.

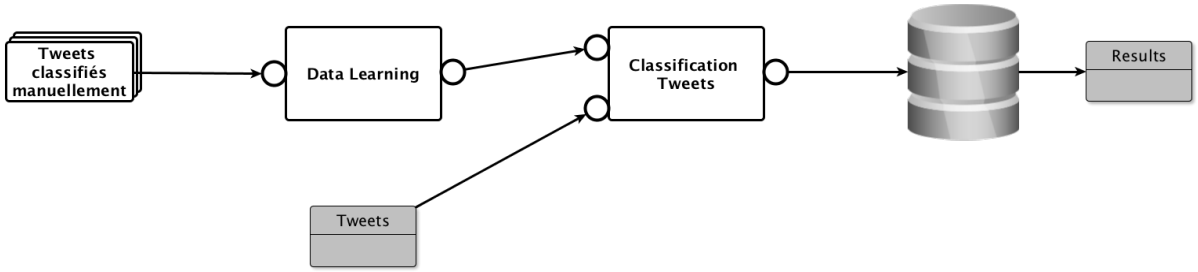

FIGURE 3.3 – Traitements pour classifier les tweets suspects

3.3 Conception ergonomique et graphique

Dans cette section, nous présentons le rendu graphique de l'application développée via des maquettes.

1 - Interface d'accueil

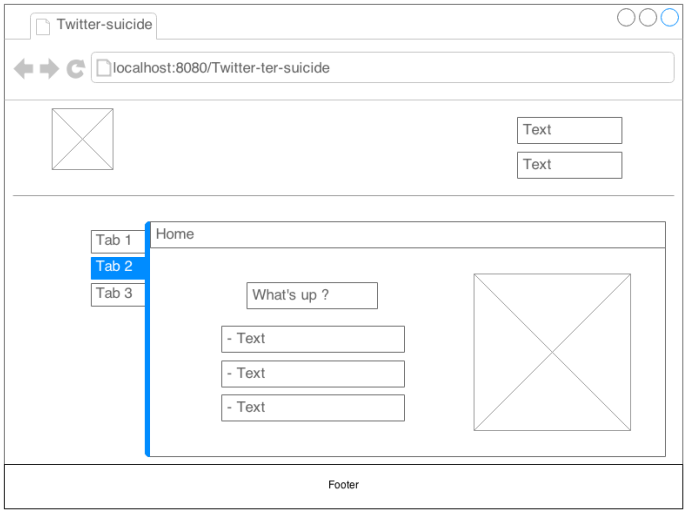

FIGURE 3.4 – Interface d'accueil

2 - Interface "Tweets suspects"

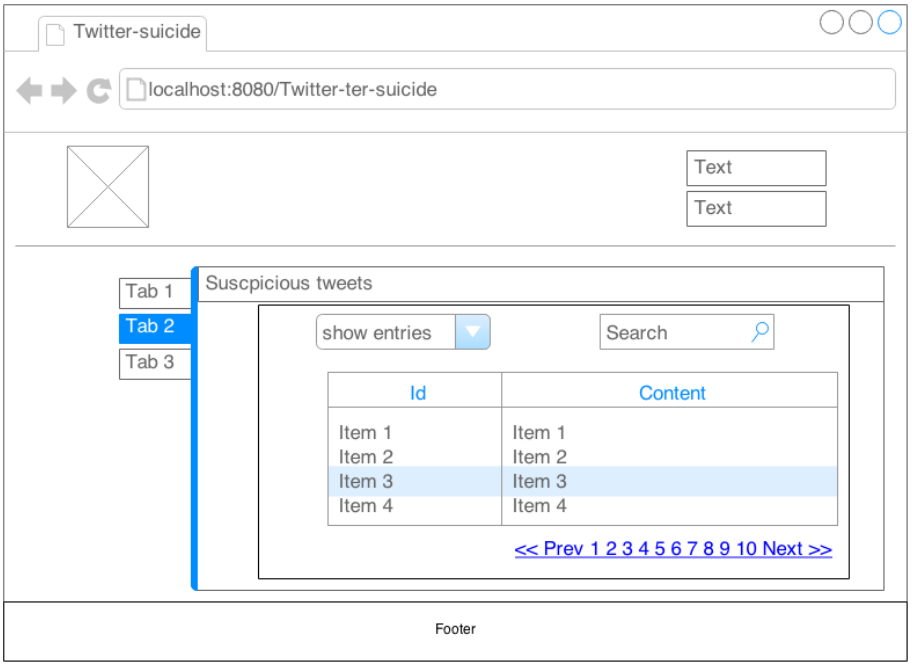

FIGURE 3.5 – Interface "Tweets suspects"

3 - Interface "Consultation des termes"

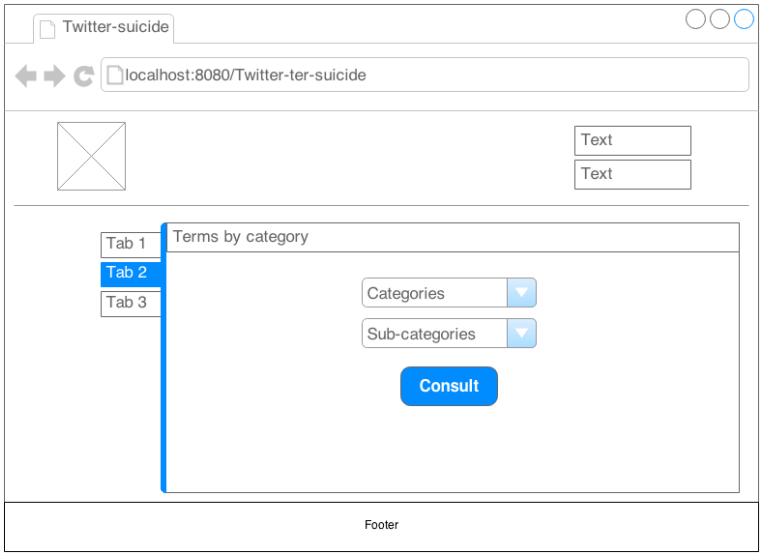

FIGURE 3.6 – Interface "Consultation des termes"

4 - Interface "Statistiques"

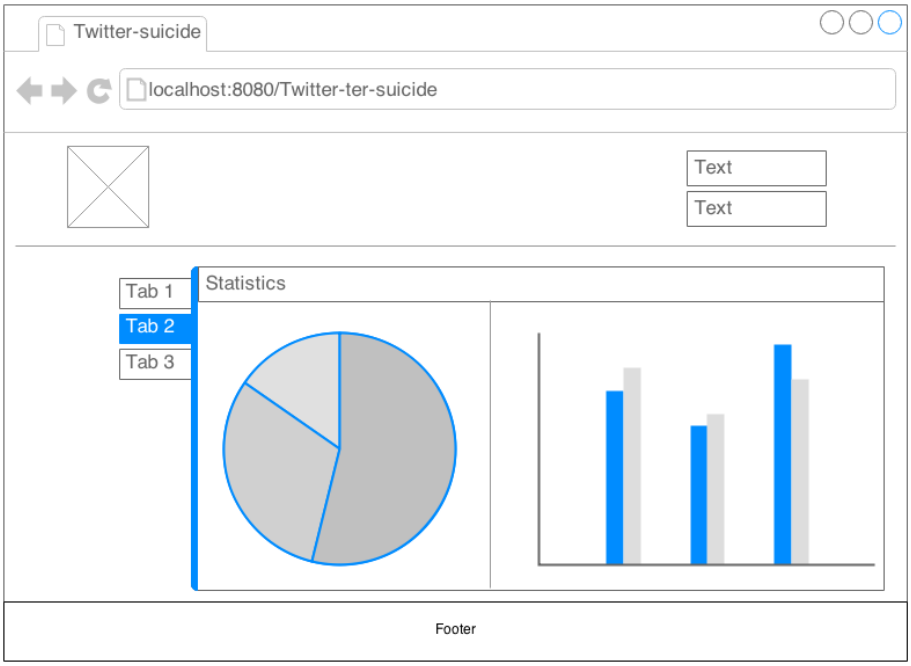

FIGURE 3.7 – Interface "Statistiques"

5 - Interface "Démonstration"

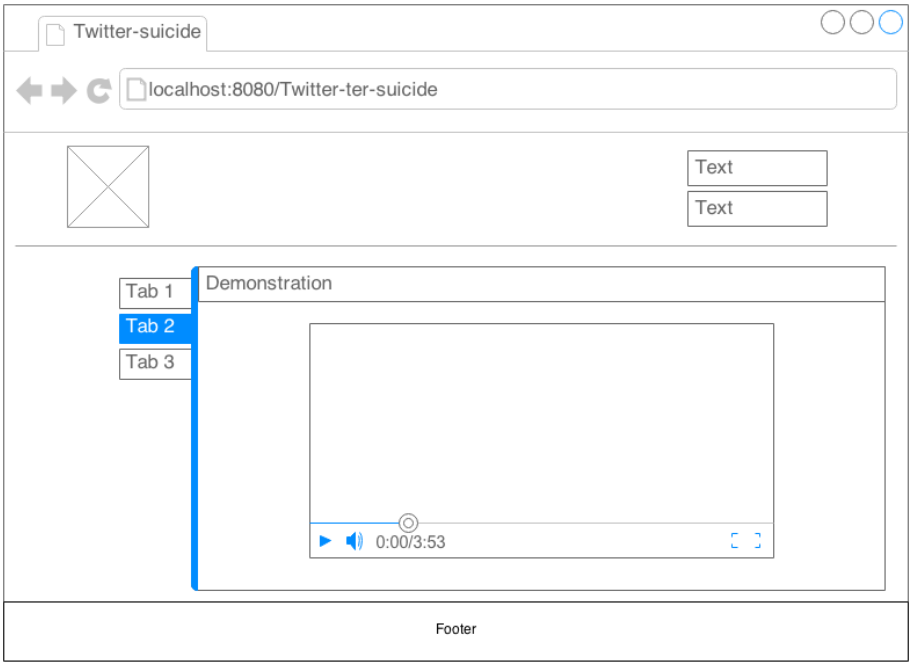

FIGURE 3.8 – Interface "Démonstration"

3.4 Schéma de la base de données

— L'application comporte deux tables principales : la table "thematics" et la table "Tweets". Le vocabulaire que nous avons construit est stocké dans la table "Thematics". Quant aux tweets suspects, nous les stockons dans la table "Tweets".

— La table "Thematics" comporte une clé primaire composée des colonnes "term" et "subcategory". Cette clé primaire permet d'avoir une unique entrée "term - subcategory".

— La table "Tweets" est identifiée par l'attribut "id_tweet". De plus, cette table présente les attributs suivants : contenu du tweet, hash_tag(s) correspondants au tweet, la date de publication, la localisation du tweet. Les différentes colonnes correspondants aux catégories (anorexie, cyberintimidation, dépression, peur, douleur, solitude, méthode de suicide et phrase) sont de type numérique et sont indispensables pour la classification.

— La table d'association "Tweets_Thematics" permet de faire la jointure entre les deux tables. Un tweet peut correspondre à un ou plusieurs termes du vocabulaire. De même, un terme du vocabulaire peut apparaitre dans un ou plusieurs tweets.

— Pour ce qui est de la table "Classification", nous y stockons les tweets que nous avons classifiés manuellement en recherchant les tweets les plus pertinents. À nouveau, cette table est indispensable pour l'outil de classification.

— La table qui représente les résultats est identifiée par l'attribut id_tweet (cf. table "tweets") ce qui permet de récupérer facilement les informations pour un tweet précis et sur l'auteur du tweet.

Le diagramme de classes représentant notre système est le suivant :

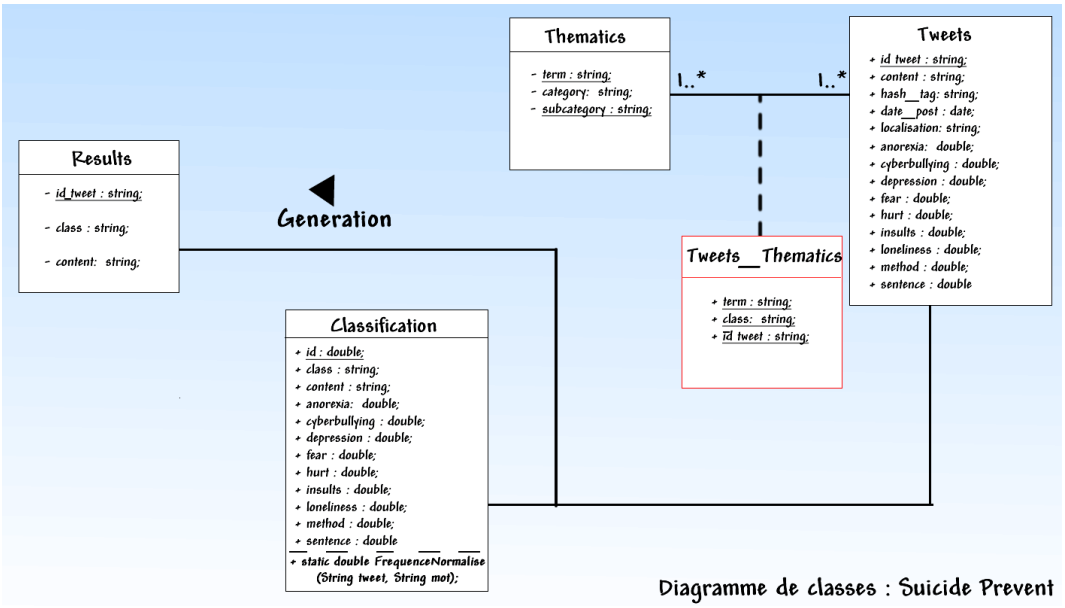

FIGURE 3.9 – Diagramme de classes

3.5 Architecture de l'application

Afin de mener à bien le développement de notre application, nous avons décidé de créer un projet permettant d'insérer les données que nous avons collectées de différentes manières (parsing de fichiers CSV pour ce qui est du vocabulaire et utilisation de l'API Twitter pour la récupération des tweets suspects) dans une base de données.
Le schéma suivant représente le processus d'insertion de ces données :

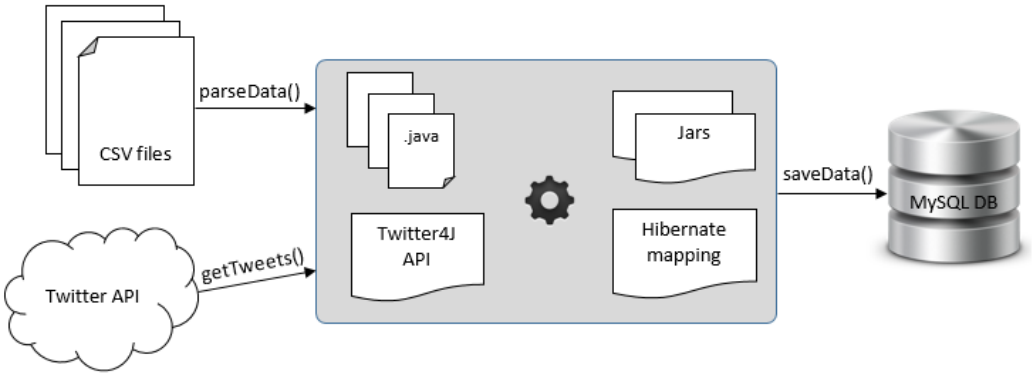

FIGURE 3.10 – Schéma détaillant l'insertion des données

Le deuxieme projet consiste en une application Java EE mettant en avant les résultats de la classification des tweets et les présentant via une interface Web.
Le schéma suivant permet de reprendre les principaux composants que nous avons utilisés :

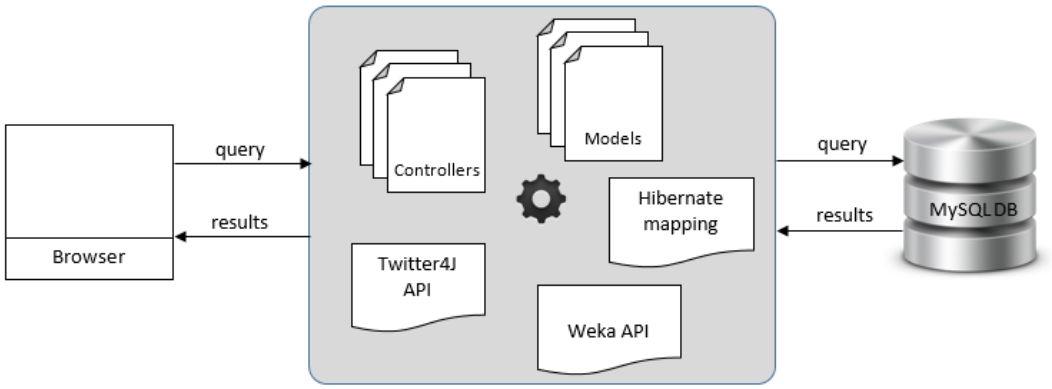

FIGURE 3.11 – Schéma de l'application Java EE

PARTIE 4 OUTILS UTILISÉS

Dans cette partie du rapport, nous présentons les principaux outils et logiciels que nous avons utilisés pour développer notre application.

4.1 Base de données

Nous avons implémenté une base de données relationnelle MySQL pour stocker le vocabulaire ainsi que les tweets suspects. Cette base de données est également utilisée comme source de données pour l'outil de classification.

Hibernate

Hibernate est une surcouche de JDBC et un framework permettant de gérer la persistance des objets Java/J2EE dans une base de données relationnelle. Le mapping objet/relationnel permet de faire le lien entre la représentation objet des données et leurs représentations dans le modèle relationnel. Concrètement, Hibernate permet de lier (mapper) un objet défini en Java avec une table dans une base de données relationnelle. Il existe deux manières de faire ce mapping :

1. Déclaration d'un fichier de mapping (format xml) ;
2. Utilisation de JPA (Java Persistance API) permettant de définir le mapping à partir d'annotations dans le code source de la classe.

Nous utilisons cette deuxième façon de faire pour le mapping entre les objets Java/J2EE et les tables de la base de données. Ceci nous a permis de peupler rapidement les différentes tables de la base de données et de traiter facilement les tweets suspects.

4.2 Collecte des messages Twitter (tweets)

4.2.1 Twitter API

Twitter expose ses données via deux API [1] bien distinctes :

1. **API Rest** : Permet de recevoir et d'envoyer des données Twitter. Cette API inclut une méthode "search" qui permet de rechercher tous les tweets comportant un certain mot-clé. Par exemple l'URL " https ://api.twitter.com/1.1/search/tweets.json ?q=%23suicide" permet de récupérer tous les tweets comportant le mot-clé "suicide". L'API propose un grand nombre de méthodes supplémentaires (il est possible, par exemple, de rechercher les tweets des personnes se situant dans une certaine zone géographique ou même de publier un tweet) ;
2. **API Streaming** : Cette API propose un flux de tweets auquel toute application peut se connecter pour récupérer un échantillon des tweets. Ainsi il est possible de récupérer un très grand nombre de tweets. Par contre, il n'est pas possible de filtrer les tweets par mot-clé comme dans l'API Rest.

1. https ://dev.twitter.com/docs

4.2.2 Twitter4J

Twitter4J [2] est une librairie Java permettant d'intégrer facilement l'API Twitter dans toute application Java. La librairie propose différentes classes et méthodes permettant de manipuler les méthodes qu'offre l'API Twitter. Pour utiliser cette librairie, il suffit de télécharger un fichier au format "jar" et de l'ajouter au classpath de l'application JAVA. La JavaDoc de la librairie permet une prise en main rapide et facile de cette librairie.

4.3 Outil de classification : API Weka

Weka [3] est un ensemble d'outils open source développé à l'université de Waikato (Nouvelle Zélande) permettant de manipuler et d'analyser des données issues de divers formats (fichiers CSV, ARFF ou encore base de données). Weka implémente un grand nombre d'algorithme de fouille de données et a été développé en Java.
L'API Weka permet d'inclure les fonctionnalités de Weka dans une application Java. Différentes classes et méthodes correspondent aux opérations que l'utilisateur peut effectuer à l'aide de l'interface utilisateur.

2. http ://twitter4j.org/en/index.html
3. www.cs.waikato.ac.nz/ml/weka/

PARTIE 5 RÉALISATION

5.1 Accès aux données

Dans cette section, nous présentons en détail les différentes approches que nous utilisons pour la manipulation des données (insertion et récupération).

5.1.1 Collecte des données

Vocabulaire associé aux thématiques critiques

L'une des premières étapes de ce projet a été d'identifier un vocabulaire associé aux thématiques critiques. Ceci fait, nous avons enregistré les différents termes au format CSV (Comma-separated values) ce qui permet de réutiliser ces données dans d'autres contextes voir d'autres applications.

Récupération des tweets

L'autre étape cruciale de notre projet est de collecter des tweets via l'API Twitter. Pour cela, nous utilisons la librairie Java Twitter4J. Comme expliqué précédemment, il y a deux types d'API Twitter, l'API Rest et l'API Streaming.
Dans le cadre de notre application, nous utilisons l'API Rest pour récupérer les tweets ayant un lien avec les différentes thématiques critiques collectées.

5.1.2 Création de la base de données

Pour les besoins de ce projet, nous avons créé une base de données implémentée sous MySQL qui nous permet d'enregistrer les tweets suspects ainsi que les termes permettant d'affirmer si un tweet est effectivement suspect ou pas.

Pour le stockage de ces données, nous utilisons le framework Hibernate qui permet de gérer la persistance des objets en base de données relationnelle.

5.1.2.1 Classes de mapping

Comme présenté dans le schéma de la base de données (section 3.4), nous utilisons deux classes de mapping Hibernate afin de manipuler nos données, la première concerne les différentes thématiques et la seconde les tweets récupérés via l'API Rest de Twitter. Ces deux classes donnent lieu à deux tables dans la base de données nommées respectivement "thematics" et "tweets".

Voici le code source de la déclaration concernant la classe de mapping "Thématiques" :

Listing 5.1 – Mapping Thématiques

```
@Entity
@Table(name = "thematics")
public class Thematics implements Serializable {

	private static final long serialVersionUID = 1L;
	@Id
	String term;
	String category;
	@Id
	@Column(name = "subcategory")
	String subcategory;

	@ManyToMany(fetch = FetchType.EAGER, mappedBy="tweetthematics",cascade = CascadeType.ALL)
	private Collection<Tweets> tweets;

	public Thematics () {

	}

	public Thematics (String term, String category, String subcategory) {
		this.term=term;
		this.category=category;
		this.subcategory=subcategory;
		this.tweets = new ArrayList<Tweets>();
	}
	/*
	* Getters and setters
	*/
}
```

La classe de mapping des "Tweets" :

Listing 5.2 – Mapping Tweets

```
@Entity
@Table(name = "tweets")
public class Tweets implements Serializable {

        private static final long
            serialVersionUID = 1L;
        @Id
        private String id_tweet;
        @Lob
        private String content;
        private String username;
        @Lob
        private String hash_tag;
        private String location;

        @ManyToMany
        @JoinTable(name="tweets_thematics",
        joinColumns=@JoinColumn(name="
            id_tweet"),
        inverseJoinColumns={
                @JoinColumn(name="term"),
                @JoinColumn(name="
                    subcategory")}
        )
        private Collection<Thematics>
            tweetthematics;

        @Column(name="hurt")
        private double hurt;
        @Column(name="loneliness")
        private double loneliness;
        @Column(name="fear")
        private double fear;
        @Column(name="depression")
        private double depression;
        @Column(name="method")
        private double method;
        @Column(name="insults")
        private double insults;
        @Column(name="anorexia")
        private double anorexia;
        @Column(name="lonely")
        private double lonely;
        @Column(name="sentence")
        private double sentence;
        @Column(name="cyberbullying")
        private double cyberbullying;

        public Tweets () {

        }

        public Tweets (String id_tweet,
            String content,String username,
            String hash_tag, String
            location)
        {
                this.id_tweet = id_tweet;
                this.content = content;
                this.username = username;
                this.hash_tag = hash_tag;
                this.location = location;
                this.tweetthematics = new
                    ArrayList<Thematics>()
                    ;
        }

        /*
        * Getters and setters
        */
}
```

5.1.2.2 Table d'association

Chaque tweet peut comporter plusieurs termes référencés dans le vocabulaire et chaque terme peut correspondre à plusieurs tweets. Afin de respecter cette relation entre les deux tables *tweets* et *thematics*, il est nécessaire d'implémenter pour chaque objet de type "Tweets" une collection de type "Thematics" et pour chaque objet de type "Thematics" une collection de type "Tweets". Ainsi, il est possible de peupler automatiquement la table d'association de la base de données et donc garder un lien entre un tweet et le terme correspondant.

5.1.3 Parsing et insertion de données

5.1.3.1 Parsing CSV

Afin de lire les fichiers portant l'extension *.csv* en Java, nous avons créé une classe utilitaire dite de *Parsing*. En effet, cette dernière contient une méthode prenant en paramètre le nom du fichier qui est chargé via les classes prédéfinies *BufferedReader* et *FileReader* que proposent le langage Java. Ceci fait, des traitements sont appliqués à chaque ligne du fichier en question afin de les stocker dans un objet bien défini puis d'ajouter chacun de ces objets dans une liste Java en l'occurence une *ArrayList* avec le type de l'objet.

5.1.3.2 Insertion dans la base de données

Grâce à Hibernate et JPA, il est très aisé de manipuler des données notamment pour ce qui est de l'insertion de ces derniers.
Nous avons créé une méthode permettant l'insertion pour chacun des types d'objets à insérer dans la base de données, que ce soit un terme ou encore un tweet suspect.
Le listing suivant montre un exemple de méthode utilisée pour insérer un ensemble de termes dans la base de données.

Listing 5.3 – Insertion d'une liste de Thématiques dans la base de données

```
public void insert_thmeaticList(ArrayList<Thematics> arrayThem)
	{
		Session session=null;
		Transaction tx=null;
		arrayThem.remove(0);
		try
		{
			session = HibernateConfig.sessionFactory.openSession();
			tx=session.getTransaction();
			tx.begin();
			for(Thematics them : arrayThem){
				System.out.println(them.getTerm());
				session.save(them);
			}
			tx.commit();
			session.flush();
		}catch(HibernateException he){
			if(tx!=null)tx.rollback();
			System.out.println("Not able to open session");
			he.printStackTrace();
		}
		catch(Exception e){
			e.printStackTrace();
		}finally{
			if(session!=null)
				session.close();
		}
	}
```

5.1.4 Chargement du vocabulaire et multi-threading

Afin d'améliorer le temps de réponse de nos requêtes SQL lors de la récupération des différents termes associés aux thématiques critiques, nous utilisons des *Threads*. En effet, cela permet d'améliorer considérablement le temps de réponse (32 secondes sans l'utilisation de threads et 12 secondes avec les threads). Cela se fait comme suit : Nous créons un thread pour chaque thématique (e.g. dépression, anorexie,...), ces derniers s'occuperont donc de charger les termes correspondants pour chacune d'elle. Ceci fait, nous stockons tous les termes récupérés dans une liste Java en vue d'un traitement ultérieur.

5.2 Classification avec Weka

Dans le but de faciliter la prise en main du logiciel Weka et de son API Java, nous avons commencé par utiliser l'interface graphique afin de découvrir les différents mécanismes qu'offre cet outil. Ceci nous a permis de faire une pré-sélection des algorithmes de classification que nous pourrions utiliser dans l'application.

5.2.1 Classification manuelle du modèle d'apprentissage

Dans un premier temps, nous avons effectué ce que nous appelons une "classification manuelle". En effet, nous avons sélectionné des tweets pour lesquels il y a une forte probabilité que leurs auteurs passent à l'acte (tweets suspects à risque). Nous avons fait de même pour les tweets sans risque. Nous avons veillé à ce qu'il y ait un équilibre entre les deux classes de tweets.
Nous avons par la suite inséré ces tweets dans la table "classification" de la base de données.
Au départ, la table classification contenait les colonnes suivantes : contenu du tweet, la sous catégorie liée au tweet ainsi que la classe du tweet ('Y' : tweet suspect avec passage à l'acte et 'N' : tweet non suspect sans intention de passer à l'acte).

Afin d'apporter plus de précision à notre modèle d'apprentissage, nous avons modifié la structure de cette table afin d'y inclure chaque catégorie (dépression, peur, anorexie, etc.) comme attribut numérique (cf. Listing 5.6).
Pour cela, nous avons créé une méthode permettant de comparer chaque mot d'un tweet à notre vocabulaire afin d'identifier la ou les catégories auxquelles appartient le tweet.
Le listing 5.4 représente le code soure de la méthode en question.

Listing 5.4 – Aperçu de la méthode de normalisation

```
public static double FrequenceNormalise(String tweet, String word){
        StringTokenizer sizeTweet = new StringTokenizer(tweet, " \r \t\n.,;:\'\"\\()?!−");
        StringTokenizer sizeWord = new StringTokenizer(word, " \r \t\n.,;:\'\"\\()?!−");
        double length  = sizeTweet.countTokens();
        double lengthmot = sizeWord.countTokens();
        int countWord = 0;
        int count = 0;
        String tw = tweet.toLowerCase();
        String wo = mot.toLowerCase();
        if (tw.contains(wo)&&lengthmot<=1)
        count++;
        else if(lengthmot>1){
        boolean test = true;
        while(sizeWord.hasMoreTokens()){
                if(!tw.contains(sizeWord.nextToken()))
                        test = false;
                else
                        countWord++;
                }
                if(test)
                        count = countWord;
        }
        return count/length;
}
```

Le schéma suivant met en évidence le fonctionnement de Weka via un exemple concret :

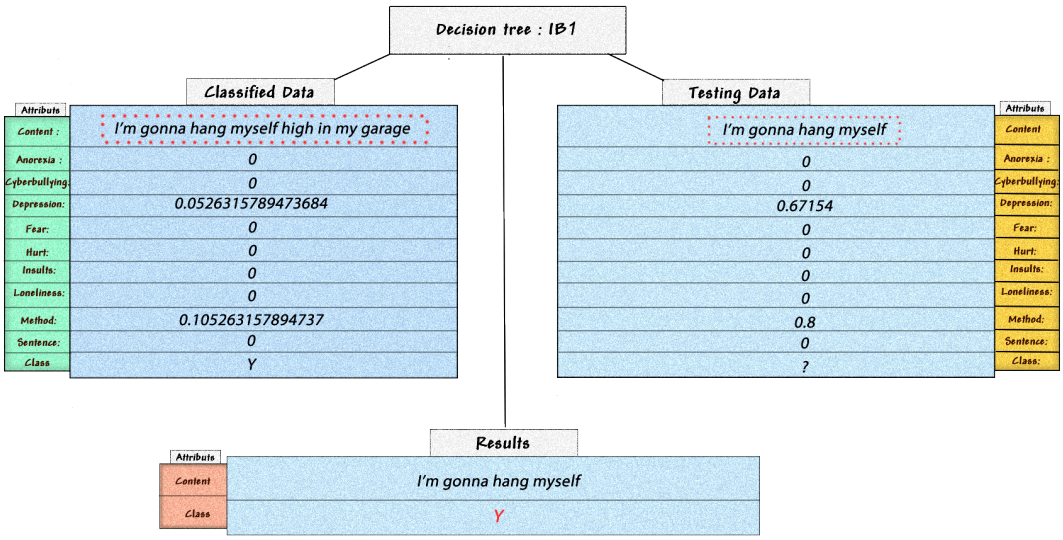

FIGURE 5.1 – Fonctionnement de Weka

5.2.2 Processus de classification Weka

Dans cette partie, nous allons présenter le processus de classification via l'API Weka.

La requête suivante permet de récupérer à partir de la BD toutes les instances permettant de construire le modèle d'apprentissage.

Listing 5.5 – Récupération des instances Weka à partir de la base de données

```
WekaConnect wekaConnect = new WekaConnect(
                                "39131_tersuicide",
                                "tersuicide",
                                "select content, anorexia, cyberbullying, depression, fear, hurt, insults" +
                                ", loneliness, lonely, method, sentence, class from classification");
```

Voici un aperçu du fichier ARFF généré :

Listing 5.6 – Fichier ARFF généré à partir de la base de données

```
@relation QueryResult
// Attributes with values
@attribute content {'My family would probably be better off without me.',
'Nausea, anxiety , may cause drowsiness, loss of apetite ??'}
@attribute anorexia numeric
@attribute cyberbullying numeric
@attribute depression numeric
@attribute fear numeric
@attribute hurt numeric
@attribute insults numeric
@attribute loneliness numeric
@attribute lonely numeric
@attribute method numeric
@attribute sentence numeric
@attribute class {y,n}
// Data Instances
@data
'My family would probably be better off without me.',0,0,0.111111,0,0,0,0,0,0,0.888889,y
'Nausea, anxiety , may cause drowsiness, loss of apetite ??',0.375,0,0.125,0.125,0,0,0,0,0,0,n
```

WekaClassify.java

Cette classe permet d'appliquer un algorithme de classification (SMO dans ce cas-là) sur un modèle d'apprentissage (DataLearning) puis d'afficher des statistiques telles que la précision, le taux d'erreurs et la matrice de confusion.

Listing 5.7 – Aperçu de la classe WekaClassify

```
public WekaClassify(Instances dataInstances) throws Exception
{
        Instances trainingData = new Instances(dataInstances);
        trainingData.setClassIndex(trainingData.numAttributes()-1);
        SMO sm = new SMO();
        sm.buildClassifier(trainingData);
```

```
        Evaluation evaluation = new Evaluation(trainingData);
        evaluation.crossValidateModel(sm, trainingData, 10, new Random(1));
        System.out.println(evaluation.toSummaryString("\nResults\n========\n",true));
        System.out.println("fMeasure :" +evaluation.fMeasure(1) +" Precision :"
                                    +evaluation.precision(1)+" Recall :" +evaluation.recall(1));
}
```

5.2.3 Tests des différents algorithmes

Le tableau ci-dessous représente les différents algorithmes de classification ainsi que les résultats que nous avons obtenu en terme de précision :

Algorithme	JRIP	IBK	IB1	J48	Naive Bayes	SMO
Précision de classification	54%	59.21%	57.89%	55.92%	63,15%	60.52%

Choix de l'algorithme utilisé

Le tableau ci-dessus montre que l'algorithme Naive Bayes possède le pourcentage de précision le plus élevé lors de son application sur le modèle d'apprentissage (DataLearning) uniquement.

Nous avons néanmoins choisi l'algorithme ***IB1*** qui nous permet d'avoir un meilleur résultat lors de l'application de ce dernier du modèle d'apprentissage sur notre jeu de données.

5.2.4 Résultats

Après avoir appliqué l'algorithme IB1 sur notre jeu de données, nous insérons les résultats dans la table "results".

Le listing suivant représente la méthode permettant d'appliquer l'algorithme IB1 sur un jeu de données en se basant sur le modèle d'apprentissage.

Listing 5.8 – Aperçu de la partie application du modèle d'apprentissage

```
public WekaModelApplying(Instances trainingData,Instances testData,Instances testDataID) throws
    Exception
{
        if (trainingData == null || testData == null){
                System.out.println("\n Please check the Training / Test database datas to do Learning\n");
                System.exit(1);
        }
        Instances learning = new Instances(trainingData);
        learning.setClassIndex(learning.numAttributes()-1);
        Instances test = new Instances(testData);
        FastVector values = new FastVector();
        values.addElement("y");
        values.addElement("n");
        test.insertAttributeAt(new Attribute("class", values), test.numAttributes());
        test.setClassIndex(test.numAttributes()-1);
        IB1 tree = new IB1();
        tree.buildClassifier(learning);
        Instances labeled = new Instances(test);
        for(int i=0; i < test.numInstances(); i++){
                double clsLabel = tree.classifyInstance(test.instance(i));
                labeled.instance(i).setClassValue(clsLabel);
```

```
        }
        BufferedWriter buffredW = new BufferedWriter(new FileWriter("weka/result_db_numeric.arff"));
        buffredW.write(labeled.toString());
        buffredW.close();
        new DBtoCSVConverter().createCSVFromARFF(labeled,testDataID,"WekaOutput/result_d_db.csv");
}
```

5.3 Interface de présentation

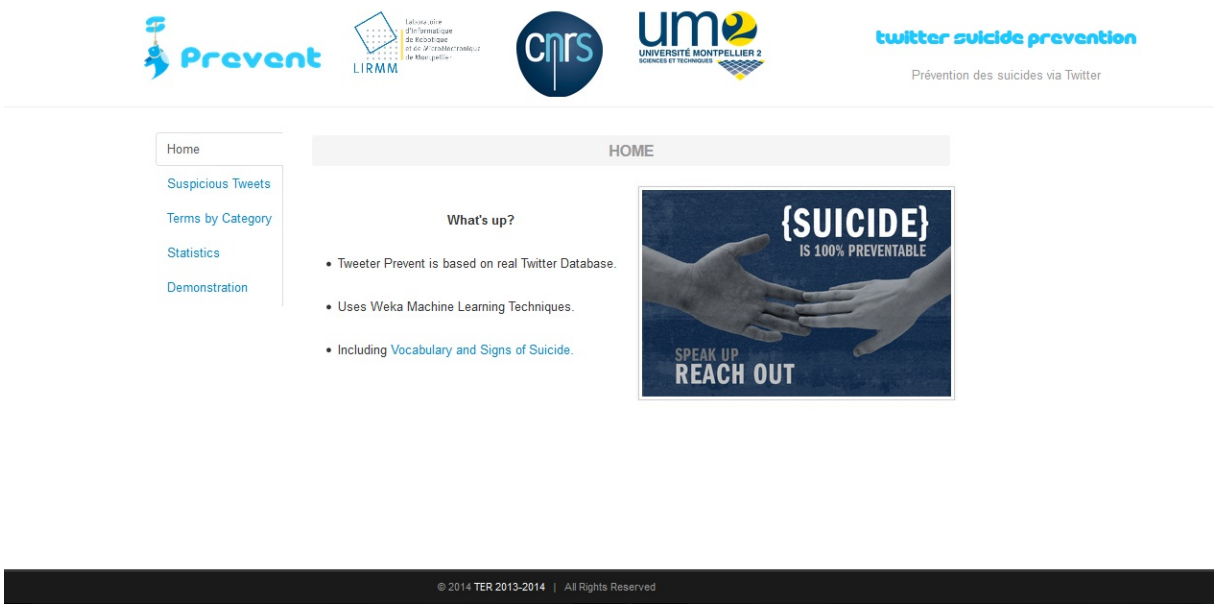

FIGURE 5.2 – Aperçu de l'interface de présentation

Nous avons développé une interface de présentation qui propose les fonctionnalités suivantes :

— Présentation du projet (Page d'accueil) ;

— Affichage et consultation des tweets suspects à risque ;

— Consultation du vocabulaire ;

— Statistiques sur les résultats et les données utilisées ;

— Visualisation d'une vidéo de démonstration.

Dans cette partie du rapport nous expliquons brièvement le fonctionnement de l'interface qui est composée de plusieurs onglets.

5.3.1 Présentation des différents onglets

5.3.1.1 Suspicious Tweets

Dans l'onglet "Suspicious Tweets" (cf. Figure 5.3), l'utilisateur peut consulter les tweets suspects à risque (résultat de la classification Weka). Une barre de recherche permet une consultation dynamique des tweets et permet par exemple de rechercher un tweet contenant un terme spécifique.

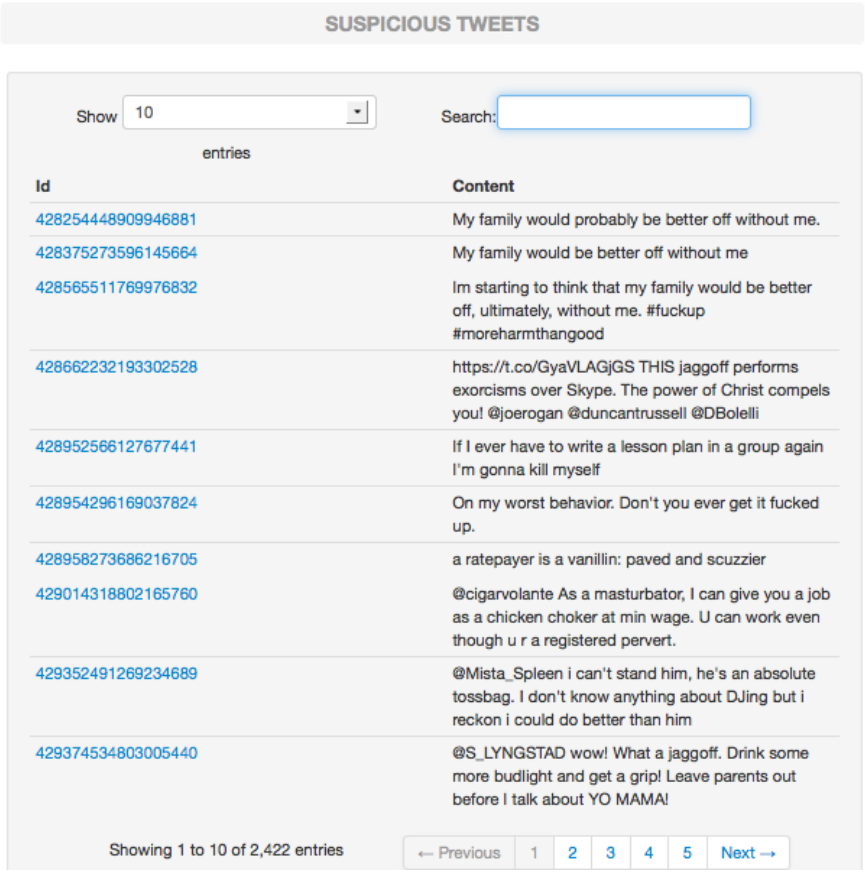

FIGURE 5.3 – Affichage des tweets suspects à risque

En cliquant sur l'identifiant d'un tweet, l'application permet à l'utilisateur de récupérer plus d'informations sur le tweet en question. Une fenêtre "popup" (cf. Figure 5.4) affiche ainsi les informations suivantes :

— Nom d'utilisateur Twitter de la personne ayant publiée le Tweet

— Nom et prénom de l'utilisateur

— Photo de profil de l'utilisateur

— Localisation de la personne ayant publiée le Tweet (si renseignée)

— Les sous-catégories du vocabulaire correspondant au tweet (permet de différencier rapidement la source de menace du tweet)

— L'historique des tweets de l'utilisateur

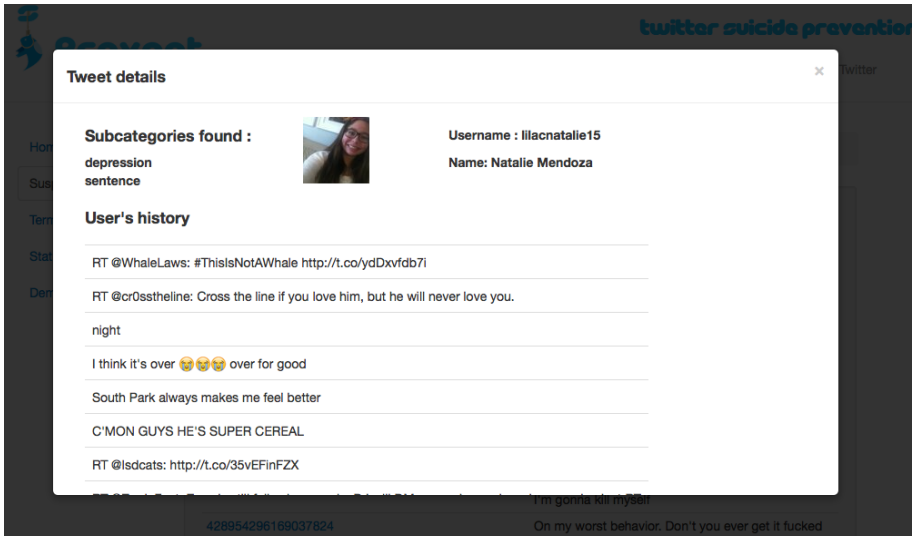

FIGURE 5.4 – Détails d'un tweet suspect à risque

5.3.1.2 Terms by category

Cette partie de l'interface de présentation permet à l'utilisateur de consulter le vocabulaire que nous avons construit. Deux listes déroulantes permettent de sélectionner la catégorie et la sous-catégorie du vocabulaire à afficher. La figure 5.5 montre l'exemple de la catégorie "Mental State" et de la sous-catégroie "Depression". À nouveau, une barre de recherche permet de rechercher un terme dans la liste.

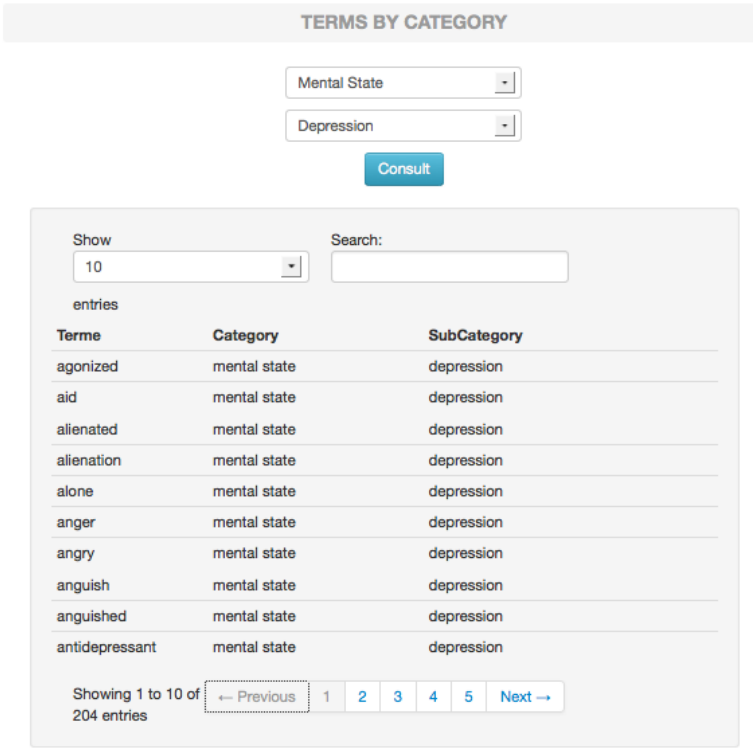

FIGURE 5.5 – Consultation du vocabulaire

5.3.1.3 Statistics

La librairie Javascript "Highcharts" [1] nous permet de présenter des statistiques par rapport aux résultats de la classification sous Weka. Deux types de statistiques sont affichées :

— Pourcentage des tweets suspects à risque par rapport aux tweets suspects sans risque (cf. Figure 5.6)

— Nombre de tweets par sous-catégorie (exprimé en pourcentage) (cf. Figure 5.7)

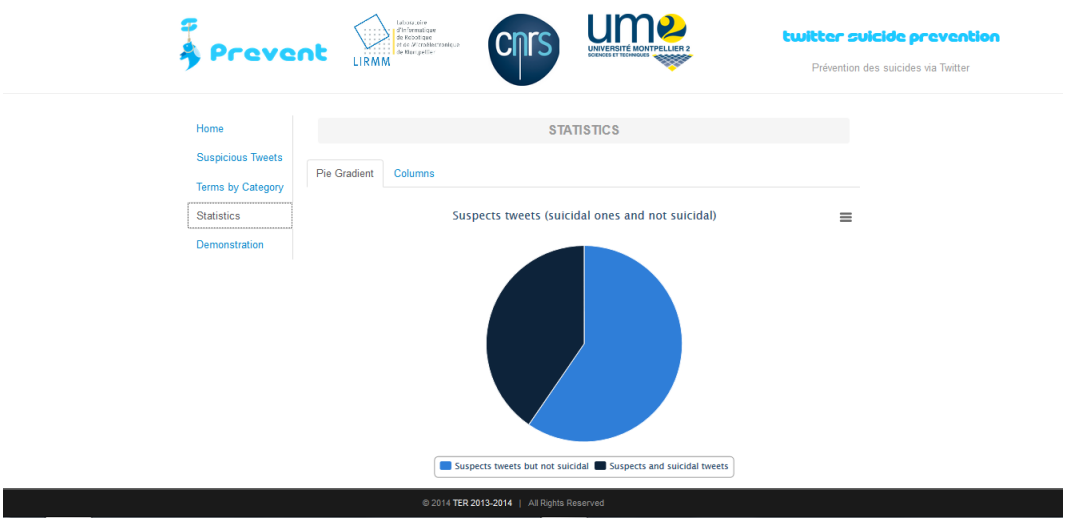

FIGURE 5.6 – Tweets à risque / Tweets sans risque

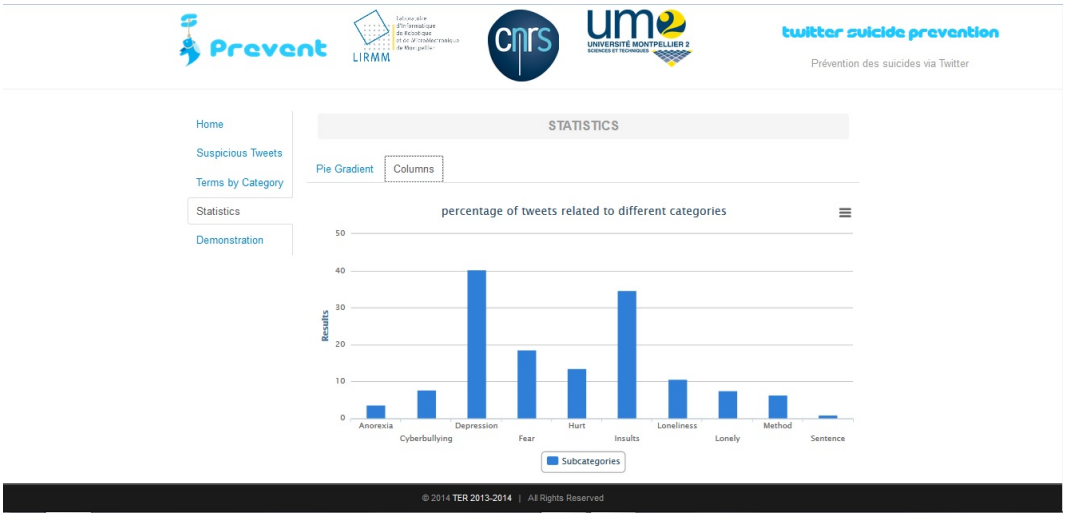

FIGURE 5.7 – Tweets suspects à risque : Pourcentage des sous-catégories retrouvées

1. http://www.highcharts.com/

5.3.1.4 Demonstration

Dans ce dernier onglet, une vidéo permet à l'utilisateur de comprendre comment utiliser l'interface de présentation ainsi que les différentes étapes que nous avons réalisées pour classifier les tweets suspects à l'aide de Weka.

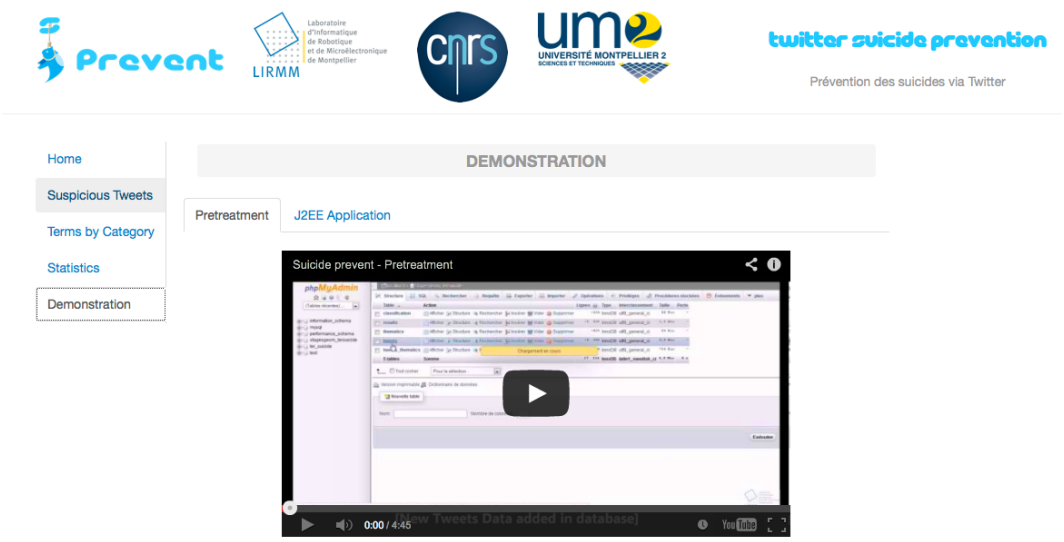

FIGURE 5.8 – Vidéo de démonstration

PARTIE 6 GESTION DU PROJET

Afin de mener à bien le projet, nous avons opté pour une gestion de projet adaptée aux exigences et aux besoins.

Nous nous sommes appuyés sur la méthodologie agile SCRUM pour chaque étape du projet ce qui nous a permis d'atteindre les objectifs de chaque sprint.

6.1 Planification du projet

• **Ressources de l'équipe :** Nous sommes une équipe de 3 analystes programmeurs : Amayas, Gilles et Yasser.

Dans un premier temps, nous avons défini les besoins et objectifs auquels le projet devait répondre. Par la suite, nous avons réalisé les étapes suivantes tout au long du développement :

• **Réunions de groupe :** Nous nous sommes réunis avec nos encadrants une fois par semaine afin de faire un point sur l'état d'avancement du projet et de planifier les nouvelles tâches à effectuer.

• **Réunions des membres de l'équipe :** Nous avons attaché beaucoup d'importance au travail collectif ce qui nous a permis un gain de temps considérable pour ce qui est de la fusion des différentes parties développées par chacun de nous.

• **Étude documentaire :** Une étude documentaire nous a permis de définir un vocabulaire associé à la thématique du suicide. Pour cela nous nous sommes basés sur plusieurs types de documents : articles scientifiques, sites web, groupes sur twitter. (cf. Bibliographie)

Dans ce qui suit, nous allons présenter les différentes étapes du projet via le concept de Sprint propre à la méthodologie Agile Scrum.

6.2 Déroulement du projet

Chaque tâche a une durée définie au préalable et un ou plusieurs membres y sont affectés.

6.2.1 Conception et étude bibliographique

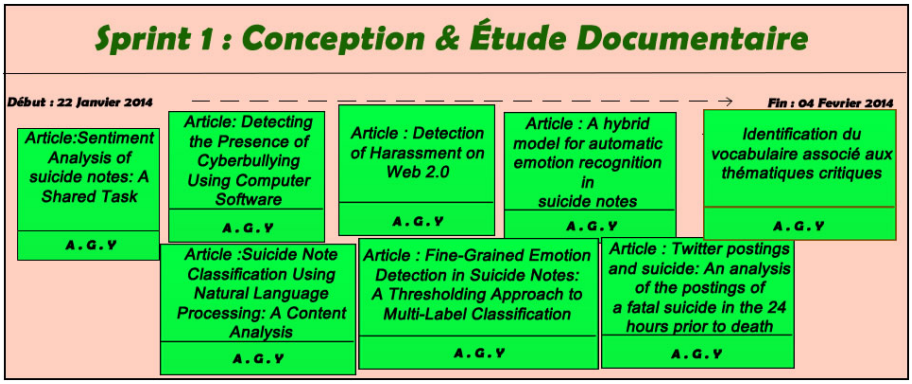

FIGURE 6.1 – Sprint1 : Conception et étude bibliographique [A : Amayas, G : Gilles, Y : Yasser]

6.2.2 Développement et programmation

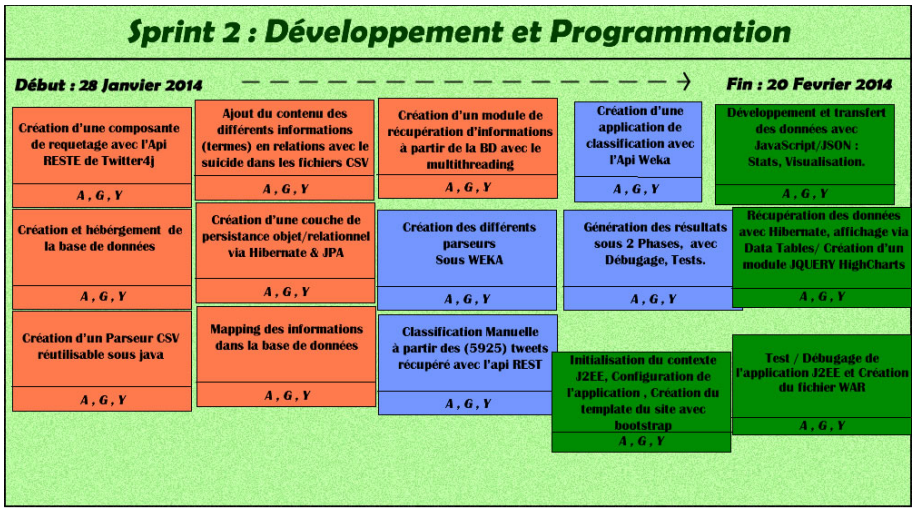

FIGURE 6.2 – Sprint2 : Développement et programmation

Pour clore cette partie, la méthodologie utilisée (SCRUM) nous a permis de mieux gérer le temps imparti.

PARTIE 7 PERSPECTIVES ET CONCLUSION

7.1 Perspectives

— Utilisation de l'API stream de twitter pour traiter le flux des données en temps réel et de prévenir les amis de l'auteur du tweet suspect en cas d'alerte ;

— Collecter plus d'informations (Tweets) d'apprentissage pour améliorer la classification automatique dans Weka ;

— Ajout de statistiques supplémentaires : Pourcentage de tweets suspects à risque et sans risque par thématique ;

— Ordonner les tweets suscpets par probabilités décroissantes du risque dans l'interface afin de faciliter l'analyse des tweets par des experts ;

— Utilisation de la classe Vote [1] de Weka qui permet de combiner plusieurs classifications en même temps afin d'améliorer l'apprentissage ;

— Ajout des connaissances basés sur les sentiments pour renforcer l'apprentissage ;

— Hébergement de l'application J2EE.

7.2 Difficultés rencontrées

Lors de la réalisation de ce projet, nous avons été confronté à plusieurs difficultés dont les principales sont les suivantes :

— Prise en main de Weka vu notre manque de connaissances dans le domaine de la fouille de données ;

— Classification manuelle des tweets selon que leurs auteurs décident de passer à l'acte ou pas ;

— Limites de l'API Twitter ;

— Projet un peu trop abitieux pour le temps imparti.

7.3 Conclusion

Ainsi, dans le cadre de ce TER du Master 2 Informatique, nous avons pu développer, en collaboration avec nos encadrants, un outil permettant potentiellement d'utiliser le réseau social Twitter comme force préventive dans la lutte contre le suicide. De plus, une interface de présentation permet à un large public de visualiser et de comprendre les traitements que nous effectuons ainsi que les résultats obtenus.

1. http ://weka.sourceforge.net/doc.dev/weka/classifiers/meta/Vote.html

Dans un premier temps, une étude documentaire nous a permis de définir un vocabulaire associé à différentes thématiques critiques liées au suicide. Ce vocabulaire a été stocké dans une base de données relationnelle MySQL.

Dans un second temps, après avoir défini les objectifs et besoins de ce projet, nous avons commencé le développement des différents traitements nécessaires pour classifier les tweets suspects.

Ce TER nous a permis de nous familiariser avec de nouvelles technologies (comme Weka ou encore la librairie HighCharts).

Pour conclure, nous espérons avoir apporté des réponses à la problématique donnée et que notre travail soit repris et approfondi.

BIBLIOGRAPHIE

BAYZICK, J., KONTOSTATHIS, A. et EDWARDS, L. (2011). Detecting the presence of cyberbullying using computer software.

GUNN, J. F. et LESTER, D. (2012). Twitter postings and suicide : An analysis of the postings of a fatal suicide in the 24 hours prior to death. *Suicidologi*.

LUYCKX, K., VAASSEN, F., PEERSMAN, C. et DAELEMANS, W. (2012). Fine-grained emotion detection in suicide notes : A thresholding approach to multi-label classification. *Biomedical Informatics Insights*.

PESTIAN, J. P., MATYKIEWICZ, P., LINN-GUST, M., SOUTH, B., UZUNER, O., WIEBE, J., COHEN, B., HURDLE, J. et BREW, C. (2012). Sentiment analysis of suicide notes : A hared task. *Biomedical Informatics Insights*.

PESTIAN, J. P., NASRALLAH, H., MATYKIEWICZ, P., BENNETT, A. et LEENAARSS, A. (2010). Suicide note classification using natural language processing : A content analysis. *Biomedical Informatics Insights*.

YANG, H., WILLIS, A., DE ROECK, A. et NUSEIBEH, B. (2012). A hybrid model for automatic emotion recognition in suicide notes. *Biomedical Informatics Insights*.

YIN, D., XUE, Z. et HONG, L. (2009). Detection of harassment on web 2.0. Rapport technique.