Rapport Projet Encadré

Présentation de l'application Tweel

Quentin BAERT

Thomas BERNARD

Récupération de l'application

Notre application Tweel est disponible sur GitHub à l'adresse :

https://github.com/Ath0m/PJE

Elle est également présente sur le dépôt SVN de l'université à cette adresse :

https://svn-etu.fil.univ-lille1.fr/svn/pje14-15-baert

Sommaire

Description générale du projet

I. Description de la problématique

Le but du projet était de créer une application capable de donner le sentiment général sur Twitter à propos d'un sujet en particulier. Le sentiment d'un tweet peut être représenté par trois classes : positif, négatif, neutre.

Afin de determiner le sentiment à propose d'un sujet, une requête à l'API Twitter est faite avec le sujet en tant que mot clef. Un certain nombre de tweets à propos du sujet sont ainsi récupérés.

Une fois les tweets récupérés, il faut alors les analyser pour les répartir dans les différentes classes et ainsi en déduire le sentiment dominant à propos du sujet.

II. Description générale de l'architecture de notre application

Les sources de l'application ont été divisées en plusieurs packages :

- **feeling** : contient l'ensemble des classes relatives aux sentiments et à l'attribution d'un sentiment à un tweet,

- **utils** : contient l'ensemble des classes utilitaires de l'application. Ces classes permettent d'ajouter une abstraction aux objets proposés par la librairie **twitter4j**. Par exemple, la classe *Tweet* fournie une abstraction d'un tweet représenté par une instance de la classe *Status* dans **twitter4j**. Ainsi une instance de *Tweet* ne contiendra que les informations nécessaires au bon fonctionnement de l'application.

- **statistics** : contient l'ensemble des classes qui permettent de faire des statistiques depuis les données de l'application. Ce package contient la classe *CrossValidation* qui permet de faire une évaluation d'un classifieur en usant de la méthode de cross validation. Il contient également la classe *PieChartBuilder* qui permet de générer les diagrammes circulaires concernant les classifications d'un classifieur sur une liste de tweets.

Les packages suivants contiennent les différents composants d'une architecture en MVC.

- **model** : contient la classe *AppModel*, modèle de l'application.

- **controller** : contient la classe *AppController*, contrôleur de l'application.

- **view** : contient l'ensemble des classes constituant la vue de l'application.
- **app** : contient la classe *Main* de l'application.

Détails des différents travaux réalisés

I. API Twitter

Pour pouvoir interroger l'API Twitter, nous avons utilisés la librairie **twitter4j**. Cette librairie donne accès au singleton *TwitterFactory*, qui permet d'accéder à l'API Twitter ainsi qu'aux classes suivantes :

- *Query* : requête
- *QueryResult* : résultat d'une requête (ensemble de *Status*)
- *Status* : tweet et toutes les informations associées

Ces classes sont uniquement utilisées dans la méthode *model.AppModel.search*. Une fois la recherche effectuée, les instances de *Status* deviennent des instances de *Tweet*. Ce sont ces instances de *Tweet* qui seront manipulées par la suite.

Afin de fixer le nombre de tweets récupérés et de s'assurer que ces tweets soient écris en alphabet latin, les opérations suivantes sont effectuées sur l'instance de *Query* utilisée pour la recherche depuis une chaine de caractères *searchQuery* :

Query query = new Query(searchQuery);

```
/* Assure le fait que les tweets récupérés soient en alphabet
latin */
query.setLang( "fr" );

/* Fixe le nombre de tweets récupérés lors de la requête */
query.setCount( this.tweetsNb );
```

II. Préparation de la base d'apprentissage

1. Nettoyage des données

Afin de nettoyer les tweets sauvegardés dans la base d'apprentissage, un singleton *MessageCleaner* est utilisé.
Ce dernier est constitué de plusieurs méthodes qui prennent une chaine de caractères et renvoient cette dernière nettoyée. Les chaines de caractères sont nettoyées à l'aide d'expressions régulières et de la méthode *String.replaceAll*.
Nettoyer le message d'un tweet revient donc à le faire passer dans chacune des méthodes du *MessageCleaner*.

Afin de pouvoir facilement ajouter une méthode de nettoyage dans le *MessageCleaner*, la réflexivité est utilisée lors de sa construction afin de construire une liste de fonctions à appliquer pour nettoyer totalement le message d'un tweet. Pour cela, une méthode chargée du nettoyage doit commencer par `delete` pour être considérée comme telle et être utilisée.

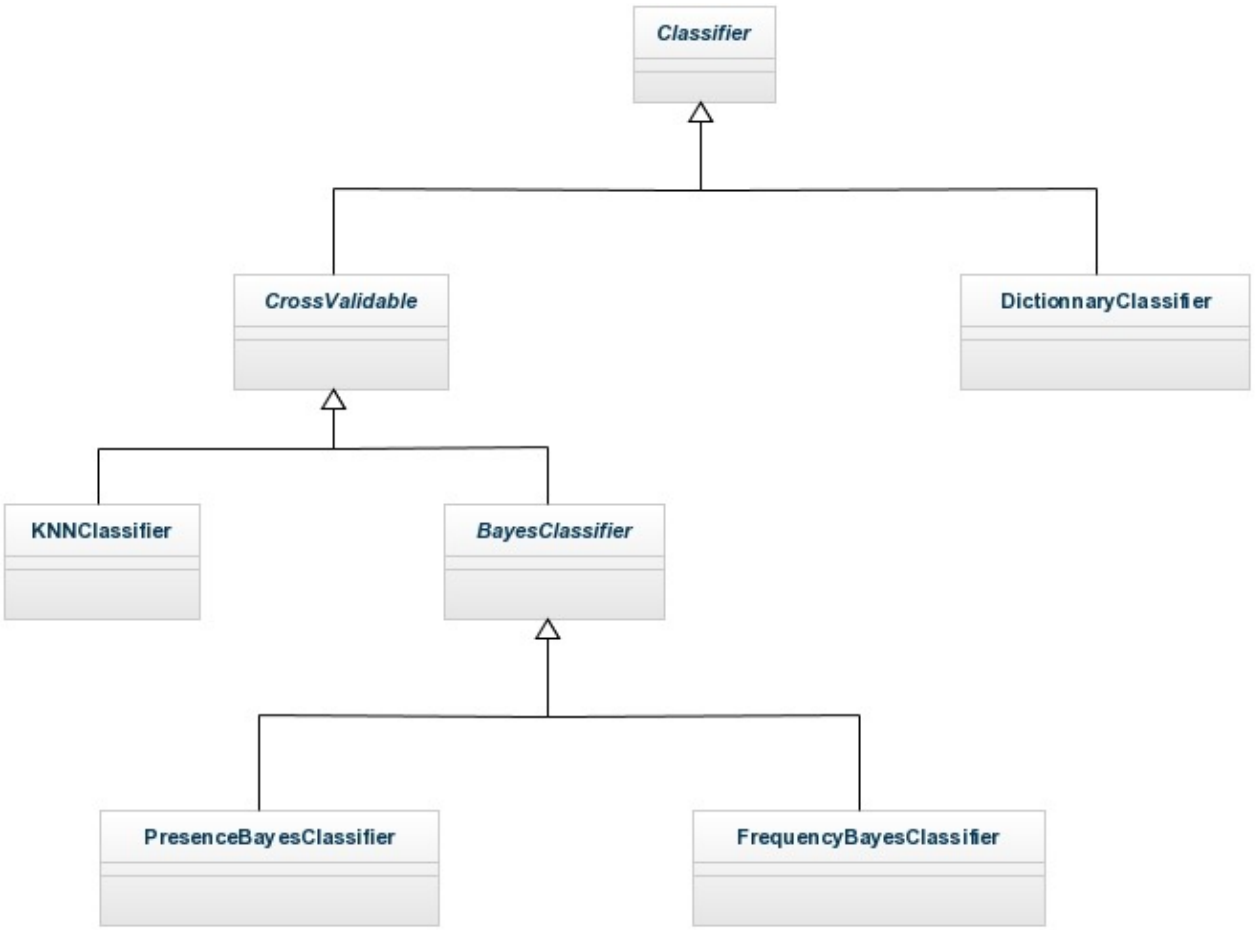
```
/* Constructeur de MessageCleaner */
private MessageCleaner () {
  this.methods = new ArrayList< Method >();

  for ( Method m : this.getClass().getMethods() ) {
    if ( m.getName().startsWith( "delete" ) ) {
      this.methods.add( m );
    }
  }
}
```

2. Construction de la base

Notre base d'apprentissage est abstraite dans une instance de la classe *TweetPool* qui correspond à un ensemble de tweets. Cette dernière est créée à l'ouverture de l'application et remplie à partir d'un fichier **tweetPool.csv** si ce dernier existe. Une fois la *TweetPool* créée, toutes les manipulations seront faites sur cette abstraction. Le fichier **tweetPool.csv** est réécrit à la fermeture de l'application pour sauvegarder le nouvel état de la base d'apprentissage.

Nous avons décidé de construire la base d'apprentissage "à la main" afin que celle-ci soit la plus précise possible. Pour cela, nous avons utilisé l'écran **Apprentissage** de notre application qui affiche des tweets obtenus depuis une requête et propose de leur associer un sentiment. Une fois ce sentiment indiqué, ces tweets sont sauvegardés dans la *TweetPool* du modèle et servent immédiatement aux classifications demandées par la suite.

III. Algorithmes de classification



Tous les classifieurs ont été regroupés sous une classe abstraite *Classifier*. On distingue ensuite les classifieurs qui peuvent être évalués par une cross validation. Dans cette catégorie se trouvent le *KNNClassifier* ainsi que les différents classifieurs se basant sur la classification bayesienne.

1. Mots clefs

Le *DictionnaryClassifier* utilise une classification par mots clefs ou par dictionnaire. Ce dernier est basé sur un algorithme simple qui utilise deux dictionnaires, un dictionnaire de mots positifs et un dictionnaire de mots négatifs. L'algorithme vérifie pour chacun des mots du tweet s'il est dans le dictionnaire positif ou négatif, si le tweet contient une majorité de mots positif il est classé comme positif, s'il contient une majorité de mots négatif il est classé comme négatif, sinon il est classé comme neutre.

2. KNN

Pour assigner un sentiment à un tweet, le *KNNClassifier* commence par trouver ses plus proches voisins selon la formule suivante :

```
distance(t1, t2) = nbMots(t1) + nbMots(t2) - (2 * nbMotsCommuns(t1, t2))
```

Dans l'application, nous avons choisis de travailler avec les 5 plus proches voisins du tweet à classifier.
Le tweet recevra le sentiment le plus représenté par ses voisins les plus proches.

3. Bayes

Les classifieurs utilisant la classification bayesienne sont basés sur une classe *NGramme* qui peut représenter des uni-grammes comme des bi-grammes. Les classes *PresenceBayesClassifier* et *FrequencyBayesClassifier* utilisent la classe *NGramme*, il suffit donc de leur donner une liste d'entiers représentant les degrés des *n-grammes* à traiter.
Ainsi :

```
List<Integer> uni = new ArrayList<Integer>();
uni.add(1);

List<Integer> bi = new ArrayList<Integer>();
bi.add(2);

List<Integer> uniBi = new ArrayList<Integer>();
uniBi.add(1);
uniBi.add(2);

/* p1 ne traitera que les uni-grammes */
BayesClassifier p1 = new PresenceBayesClassifier(..., ..., uni);

/* p2 ne traitera que les bi-grammes */
BayesClassifier p2 = new PresenceBayesClassifier(..., ..., bi);

/* p3 traitera les uni-grammes et les bi-grammes */
BayesClassifier p3 = new PresenceBayesClassifier(..., ..., uniBi);
```

La classification bayesienne est une classification probabiliste qui, selon les cas, utilise la fréquence ou la présence des *n-grammes* issus du tweet à classifier.

IV. Interface graphique

L'interface de notre application se décompose en deux éléments.
A gauche, une barre latérale fait office de menu et permet de naviguer entre les différentes fonctionnalités de l'application. A droite, la partie principale de la fenêtre, représente les différentes fonctionnalités liées à l'analyse de sentiments sur Twitter.

Dans le détail, le menu permet d'accéder à :

- **Sentiments** :

Cet écran permet d'effectuer une recherche sur un thème particulier, d'obtenir les tweets correspondants à cette recherche et d'indiquer pour chaque tweet le sentiment déterminé par nos algorithmes.

- **Tendances** :

Comme pour la section **Sentiments**, cet écran permet d'effectuer une recherche sur un thème particulier mais cette fois les tweets ne sont pas individuellement affichés. En revanche, une estimation du sentiments des tweets trouvés est effectuée puis affichée sous forme de diagramme circulaire.

- **Apprentissage** :

Cet écran permet d'effectuer une recherche sur un thème particulier et de définir manuellement un sentiment sur un tweet. Le sentiment déterminé par notre application est indiqué à titre de comparaison, mais c'est à l'utilisateur de choisir le sentiment approprié, dans le but d'ajouter ce tweet à notre base d'apprentissage et améliorer les algorithmes de classifications.

- **Evaluation** :

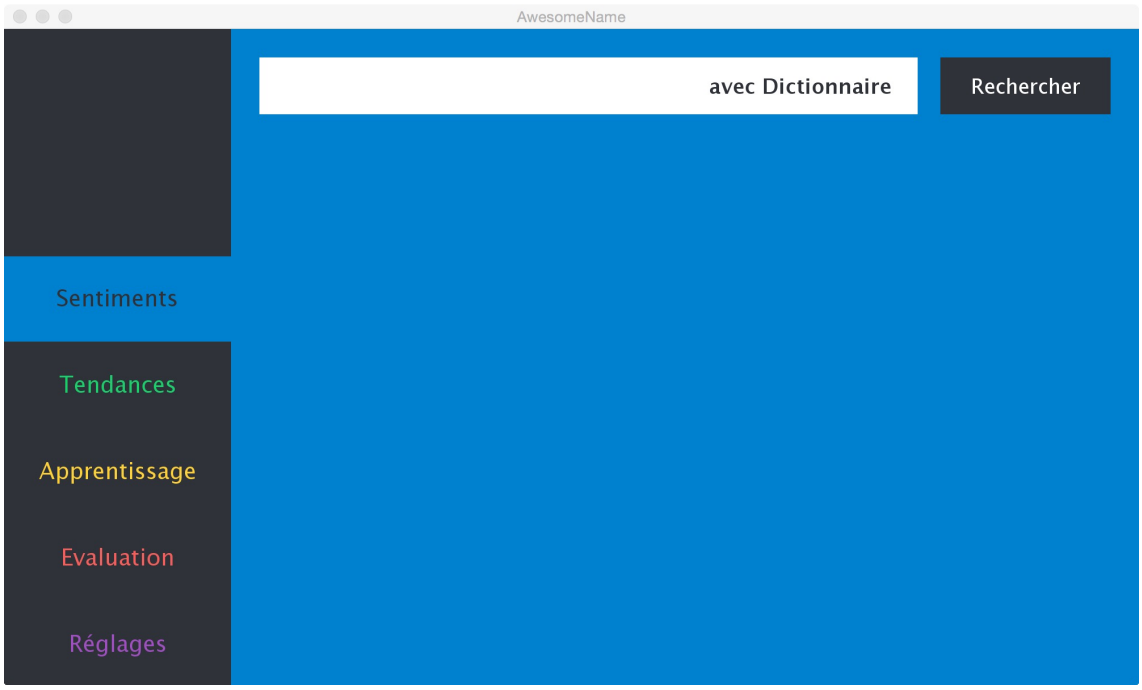
Cet écran permet d'évaluer la qualité du classifieur utilisé par l'application. Le classifieur est utilisé sur l'ensemble de la base d'apprentissage pour déterminer le taux d'erreur de celui-ci.

- **Réglages** :

Cet écran permet de modifier le comportement de l'application. Trois paramètres sont ainsi configurables :

  - *Classifieur* : Permet de choisir le classifieur à utiliser dans le reste de l'application
  - *Nombre de tweets* : Permet de choisir le nombre de tweets à utiliser/analyser dans le reste de l'application
  - *Proxy Lille 1* : Permet d'activer ou non l'utilisation du proxy Lille 1 pour l'accès à l'API Twitter

1. Copies d'écrans



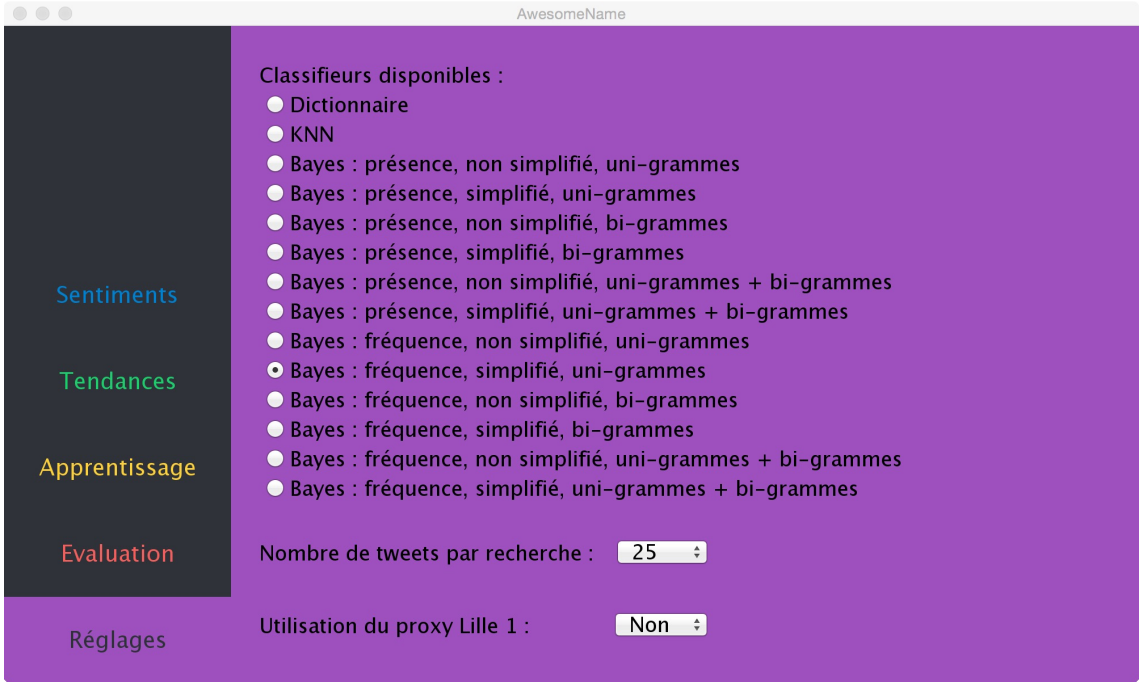
Page d'accueil de l'application

Recherche de tweets sur le mot-clé "Lille", et affichage du sentiment déterminé à partir du classifieur Dictionnaire

Recherche d'une tendance sur le mot-clé "Lille", et affichage d'un diagramme circulaire donnant la répartition entre tweets positifs, négatifs et neutres, à partir du classifieur Bayes par présence

Recherche de tweets sur le mot-clé "Lille", affichage des résultats avec indication du sentiment à l'aide du classifieur Bayes par présence. Les différents boutons sous chaque tweet permettent de lui associer un sentiment et de l'ajouter à la base d'apprentissage

Evaluation du classifieur actuellement utilisé par l'application, c'est à dire Bayes par fréquence, et calcul en temps réel de son taux d'erreur



Paramètres de l'application, avec choix du classifieur à utiliser, du nombre de tweets à traiter simultanément et de l'utilisation du proxy Lille 1

2. Manuel d'utilisation

/!\ L'application Tweel nécessite Java 7 ou ultérieur

L'application Tweel est fournie sous la forme d'un fichier JAR.

Ce fichier JAR s'accompagne d'un dossier `resources` et d'un fichier `twitter4j.properties`. **Ces deux éléments sont indispensables au bon fonctionnement de l'application et doivent être présent dans le même dossier que tweel.jar.**

- `twitter4j.properties` contient les tokens d'authentifications à l'API Twitter. En l'état, l'application peut être utilisée avec les accès fournis mais vous pouvez utiliser des tokens différents.

- `resources` est un dossier contenant les fichiers :
  - `negative.txt` est utilisé par le classifieur Dictionnaire pour identifier les mots à connotation négative.
  - `positive.txt` est utilisé par le classifieur Dictionnaire pour identifier les mots à connotation positive.
  - `tweetPool.csv` est le fichier qui contient la base d'apprentissage. Celui-ci se met à jour automatiquement lorsque vous notez manuellement des tweets dans la section Apprentissage de l'application.

Exécution de l'application :

```
java -jar tweel.jar
```

La section **Sentiments** de l'application permet de connaitre le sentiment de chaque tweet pour une recherche donnée.
Entrez le terme recherché dans le champ en haut de l'écran et lancez la recherche à partir du bouton *Rechercher*. Les tweets correspondants s'affichent, avec l'identifiant, le contenu du tweet, et l'évaluation de son sentiment. Ce sentiment prend la forme d'un **+** pour les tweets **positifs**, un **-** pour les tweets **négatifs**, un **=** pour les tweets considérés comme **neutres**.

La section **Tendances** de l'application permet de connaitre le sentiment général pour une recherche donnée.
Entrez le terme recherché dans le champ en haut de l'écran et lancez la recherche à partir du bouton *Rechercher*. Les tweets correspondants sont analysés et sont utilisés pour générer un **diagramme circulaire** indiquant la proportion de tweets jugés **positifs**, **négatifs** et **neutres**.

La section **Apprentissage** de l'application permet d'améliorer les résultats globaux de l'application en annotant manuellement des tweets.
Entrez le terme recherché dans le champ en haut de l'écran et lancez la recherche à partir du bouton *Rechercher*. Les tweets correspondants s'affichent, avec l'identifiant, le contenu du tweet, et l'évaluation de son sentiment. Ce dernier est une indication sur le sentiment

estimé dans l'état actuel de la base d'apprentissage. Vous pouvez, à partir des quatre boutons présents sous le tweet, **définir manuellement le sentiment du tweet**. Il n'est pas obligatoire de noter tous les tweets de la liste. Une notation est immédiatement enregistrée dans la base d'apprentissage de l'application.

Le comportement de ces trois sections peut être configuré dans l'onglet **Réglages** de l'application.
Sur cet écran, il est possible de **choisir un classifieur différent pour évaluer les tweets**. Le changement de classifieur prend effet immédiatement. De plus, votre choix est enregistré et conservé entre deux utilisations de l'application.
Il est également possible de définir le **nombre de tweets** attendus lors d'une recherche. Ce critère sera notamment utilisé pour calculer le diagramme circulaire de la section **Tendances**.
Finalement, une option permet d'activer ou non l'utilisation du **proxy Lille 1** pour interroger l'API Twitter à partir des ordinateurs de l'université.

La section **Evaluation** de l'application permet d'évaluer la qualité des classifieurs disponibles.
L'évaluation s'effectue sur le classifieur actuellement choisi dans les **Réglages**. Le bouton *Evaluer* lance l'évaluation. **Cette opération peut prendre un certain temps et dépend de la taille de la base d'apprentissage.** Une fois le taux d'erreur calculé, celui-ci est affiché sous la forme d'un pourcentage.

Résultats de la classification

- L'évaluation du *DictionnaryClassifier* n'a pas été implémentée. Nous supposons cependant que son efficacité est totalement liée à la perspicacité et à la complétude de ses deux dictionnaires.

Pour les classifieurs *CrossValidable*, la méthode d'évaluation de cross validation est utilisée.

- L'évaluation du *KNNClassifier* est globalement mauvaise avec un taux d'erreur supérieur à 60%.

La fonction de distance entre deux tweets en est peut être responsable étant donné que des tweets très courts (un ou deux mots) minimiseront parfois plus la distance que des tweets contenant beaucoup de mots en communs.

- L'évaluation des *BayesClassifier* donne un taux d'erreur inférieur à 50%.

Les meilleurs classifieurs implémentés sont donc les *BayesClassifier* avec un peu plus d'une classification correcte sur deux. Parmi eux, ce sont les *BayesClassifier* basés sur les uni-grammes qui s'avèrent être les plus efficaces.
Il nous semble tout de même intéressant de revoir les algorithmes de classifications bayesiennes pour obtenir de meilleurs résultats.
On pourrait également se pencher sur l'amélioration des dictionnaires du *DictionnaryClassifier* et de la fonction de distance du *KNNClassifier*.

Conclusion

Nous avons rencontrés beaucoup de difficultés durant l'élaboration de cette application : tout d'abord dans sa simple réalisation en essayant de la rendre modulable, cohérente et en refusant de faire des traitement dans la vue. Mais surtout dans l'écriture des différents algorithmes de classification qui restent compliqués à valider et dont les résultats sont difficilement justifiables.

Cependant, la conception de cette application a été une expérience plutôt agréable pour nous. Elle nous a en effet permis de developper une application complète et utile, basée sur des algorithmes complexes et en totale autonomie. Nous avons également mis en place une architecture MVC fonctionnelle.

Nous sommes également conscients que l'application est entièrement perfectible dans les détails. Une idée de fonctionnalité à ajouter serait par exemple de pouvoir manipuler plusieurs bases d'apprentissages distinctes et de choisir l'une d'entres elles dans les réglages pour tester rapidement les classifieurs sur des échantillons différents.