Rapport du TER GMIN401 :
Intégration et optimisation d'algorithmes de classifications supervisées pour Weka

Par : ALIJATE Mehdi - NEGROS Hadrien - TURKI Batoul

31 Janvier 2014

Table des matières

Résumé

Ce sujet vise à intégrer et à optimiser des algorithmes de classifications supervisées de documents dans la suite logiciel WEKA. Ces algorithmes sont issus de travaux de recherche menés récemment au sein du LIRMM.

1 Introduction

La classification de documents est le mécanisme consistant à classer automatiquement des ressources la classe prédéfinie lui correspondant le mieux.
Plusieurs formes de classification existent (par genre, par opinion, par thème...etc), et se font via des algorithmes de classifications spécifiques. Ceux-ci se basent sur des méthodes principalement numériques (probabilistes), avec des algorithmes utilisant les mathématiques ou basés sur la recherche d'information.
Ce TER vise justement à intégrer des algorithmes de classifications supervisées de documents dans la suite logiciel WEKA [1], se basant sur un nouveau modèle de classification à partir d'un faible nombre de document, intégrant de nouvelles pondérations adaptées.
Tout d'abord, il faudra explorer l'API de WEKA, pour prendre en main du code source, la maniabilité des classes et explorer une méthode d'ajout d'un algorithme de classification. Ensuite, nous nous pencherons sur le développement des différentes classes en établissant une méthodologie concrétisant le travail mené au laboratoire du LIRMM, s'en suivra une phase d'intégration et de tests.

2 Exploration de WEKA

Après la réunion du 24/01/14, nous avons établi un plan de travail pour bien mener et répartir les tâches de ce TER. Il a été décidé de le diviser en trois grandes parties. La première, qui est décrite ci-dessous consiste à explorer et prendre en main l'API de WEKA, afin de pouvoir y rajouter les algorithmes que l'on aura développé lors de la deuxième partie, et qui seront testés et intégrés lors de la troisième.

2.1 L'API Weka/Sources avec Eclipse

Pour explorer l'API, nous nous sommes aidés de l'IDE Eclipse, qui permet facilement parcourir les sources d'une librairie externe. Après avoir étudié l'arborescence des classes de l'API, nous avons pu cibler les différentes classes et méthodes qui nous intéressent, et étudié leurs fonctionnement. Nous nous sommes aidé de ce wiki [2].

2.2 L'utilisation des classes

Une fois familiarisés avec l'API Weka, on a creusé un peu plus du côté des classes qui pourraient nous être utiles pour ce TER. Il s'agit des certaines classes présentes dans le package "weka.classifiers". En effet, notre but étant d'intégrer des algorithmes de classification, il est utile de savoir comment tournent les algorithmes de classifications, leur paramétrage et l'architecture pour organiser les ressources pour ces derniers.
Quelques tests ont été menés notamment pour bayes naif multinomial, que nous avons fait tourné sur différentes données, et avec différentes options.

2.3 Ajout d'un algorithme dans Weka

Après avoir étudié en détail la classe *NaiveBayesMultinomial*, nous avons remarqué que le calcul des pondérations (dans l'implémentation de Weka, seul la mesure intra-classe Tf est utilisée) se fait dans la méthode **buildClassifier**. Nous allons donc créer une sous classe de *NaiveBayesMultinomial*, contenant une méthode surchargeant **buildClassifier** dans laquelle nous calculerons toutes les pondérations supplémentaires.
Une fois tout cela creusé et vu en détails, il faudra intégrer l'algorithme dans l'écosystème de Weka, c'est à dire, pour le rendre disponible dans l'Explorateur, expérimentateur, etc . Weka prend en charge les classes dérivées dans le package, ceci est géré par le *GenericPropertiesCreator*. Il faudra donc dire à Weka où trouver notre nouveau classificateur et il s'occupera de l'afficher dans la *GenericObjectEditor*.

1. Weka est une suite populaire de logiciels d'apprentissage automatique. Écrite en Java, développée à l'université de Waikato, Nouvelle-Zélande. Weka est un Logiciel libre disponible sous la Licence publique générale GNU.
2. http://weka.wikispaces.com/

Nous y reviendrons plus en détails lors de la troisième étape de notre TER : L'intégration des algorithmes dans WEKA.

3 Des nouvelles méthodes de classification

3.1 Pondérations intra-classes

//TODO

3.2 Pondérations inter-classes

//TODO

3.3 Algorithmes de classifications

//TODO

4 Développement des différentes classes

4.1 Méthodologie

//TODO

4.2 Extension de Naive Bayes Multinomial

//TODO

4.3 Class-Feature-Centroide

//TODO

5 Intégration et tests

5.1 Intégration dans l'écosystème de Weka

Une fois nos classes, et donc nos trois algorithmes implémenter en langage Java (Cf. Chapitre 4) :

- `NaiveBayesMultinomialTER.java` : construit le modèle de classification avec les quatre pondérations définies (le choix de la pondération se fait via les options).
- `NaiveBayesMultinomialTERab.java` : construit le modèle de classification avec les quatre pondérations définies en variant les valeurs de a : α et b : β (le choix des valeurs de α et β se fait via les options).
- `CFCTERab.java` : construit le modèle de classification Class-Feature-Centroide en variant les valeurs de a : α et b : β (le choix des valeurs de α et β se fait via les options).

Une fois nos trois .java prêts, l'intégration dans Weka est prise en charge via *GenericPropertiesCreator*. C'est là où on peut dire à Weka où trouver nos nouveaux classifieurs et il s'occupera de les afficher dans *GenericObjectEditor*, et donc dans l'interface graphique. La procédure détaillée est :

Prérequis :

- ANT : logiciel créé par la fondation Apache qui vise à automatiser l'opération de construction d'un JAR. [3]
- JDK

3. http://ant.apache.org/

— Weka (version 3.6.10 dans notre cas)

Préparation de Weka

À l'aide d'un gestionnaire d'archives, il faudra désarchiver weka-src.jar, qui se trouve dans le répertoire de Weka une fois ce dernier installer. Ceci vous donne accès aux différents répertoires et sources du logiciel.
NB : Pour éviter toutes confusions ou conflits, il est préférable de créer un dossier *temp* par exemple, et d'y mettre votre répertoire *weka-src*.

Ajout des nouveaux classifieurs dans Weka

À ce stade du processus d'ajout des algorithmes dans Weka, il faut modifier le fichier GenericObjectEditor.props (non le .java, se trouvant au même répertoire), qui se trouve dans
`/temp/weka/weka-src/src/java/weka/gui/`, en y ajoutant les trois lignes suivantes :

```
weka.classifier.bayes.CFCTERab,\
weka.classifier.bayes.NaiveBayesMultinomialTER,\
weka.classifier.bayes.NaiveBayesMultinomialTERab,\
```

et ce, en respectant l'ordre alphabétique défini dans le fichier, et surtout au bon endroit (dans la liste après le marquage suivant : `# Lists the Classifiers I want to choose from`).
Ensuite, il suffit de placer les trois fichiers .java développés dans le répertoire :
`/temp/weka/weka-src/src/java/weka/classifiers/bayes/`.

Reconstruction de Weka

Comme son nom l'indique bien, cette dernière étape permet de reconstruire Weka avec ses nouvelles propriétés. Il suffit, après avoir installé ANT(voir prérequis ci-dessus), de se placer dans le répertoire
`/temp/weka/weka-src/`, à l'aide d'un terminal, et lancer la commande suivante :

```
ant exejar
```

Celle-ci fera appel automatiquement au fichier `build.xml`, qui, comme expliqué dans la sous-section 2.3, construira de nouveau Weka, et donnera en sortie dans le répertoire `/temp/weka/weka-src/dist/` un nouvel executable `weka.jar`, contenant les nouveaux algorithmes de classifications.

5.2 Tests

Afin de tester la pertinence de nos nouveaux algorithmes de classifications, nous avons créer 2 fichiers `.arff`, avec deux jeux de données aléatoires mais cohérents.

— `test3classes.arff` : **150** instances et **41** attributs (une selection d'attributs a été faite dessus), avec **3** classes : Policier, Fantastique, Comédie.
— `test5classes.arff` : **248** instances et **5082** attributs au complet (sans selection d'attributs), avec **5** classes : Thriller, Western, Guerre, Policier, Sciences.

Le but étant de classifié les films selon leur catégorie cinématographique . Nos deux fichiers .arff ont subi de multiples tests avec nos trois algorithmes, avec différentes valeurs pour nos variables α et β, que ce soit pour le NaiveBayesMultinomialTERab ou le CFCTERab.
Les résultats des tests sont donnés dans la section ci-dessous.

5.3 Résultats

Les deux tableaux suivants montrent les résultats relevés suite aux expérimentations effectuées avec nos trois algorithmes sur nos deux fichiers de tests.

NBMultinomialTER/fichierTest	$Nb^{Tf-Class}$	$Nb^{Df-Class}$	Nb^{Tf-Doc}	Nb^{Df-Doc}	NBMultinomial
test3classes.arff	66%	**67%**	64%	66%	66%
test5classes.arff	52%	**68%**	51%	50%	63%

TABLE 1 – Expérimentations avec les quatre pondérations et comparaison avec NBMultinomial

Le tableau 1 présente les résultats de classifications correctes atteintes avec distinctement les quatre pondérations $Nb^{Tf-Class}$, $Nb^{Df-Class}$, Nb^{Tf-Doc} et Nb^{Df-Doc} de notre algorithme `NaiveBayesMultinomialTER`.

Nous constatons que l'utilisation de la pondération $W^{Df-Class}$, pour nos deux jeux de données, donne des résultats supérieurs à ceux d'une classification avec NaïveBayesMultinomial.

Algo/FichierTest	α	β	NBMTERαβ	CFCTERαβ	NBMultinomial
test3classes.arff	α= 0.0	β= 1.0	**67%**	68%	66%
	α= 0.6	β= 0.6	66%	**74%**	
	α= 0.7	β= 0.3	66%	**73%**	
test5classes.arff	α= 0.0	β= 1.0	**67%**	**68%**	63%
	α= 0.6	β= 0.6	65%	**70%**	
	α= 0.7	β= 0.3	58%	60%	

TABLE 2 – Expérimentations avec différentes valeurs de α et β pour NBTERαβ et CFCTERαβ

6 Discussion et Conclusion

//TODO

7 Sources

//TODO