Apprentissage automatique et contraintes

1 Introduction

Dans cette dernière partie nous verrons comment combiner la satisfaction de contraintes et l'optimisation sous contraintes avec de l'apprentissage automatique.

2 Retour sur l'apprentissage automatique

Nous utiliserons le logiciel Weka pour les expérimentations. Weka est un ensemble de classes et d'algorithmes en Java implémentant les principaux algorithmes de data mining. Il est disponible gratuitement à l'adresse `http://www.cs.waikato.ac.nz/ml/weka/`, dans des versions pour Unix et Windows. Ce logiciel est développé en parallèle avec un livre : Data Mining par I. Witten et E. Frank (éditions Morgan Kaufmann). Weka peut s'utiliser de plusieurs façons :
– par l'intermédiaire d'une interface utilisateur, idéal pour explorer les données.
– sur la ligne de commande, plus adaptée à des données volumineuses.
– par l'utilisation des classes fournies à l'intérieur de programmes Java.
Télécharger le logiciel sur le site et installer le. Vous pouvez ensuite lancer son interface graphique avec la commande :

```
java -jar /chemin/vers/weka.jar
```

Après l'avoir lancé, vous obtenez la fenêtre intitulée Weka GUI Chooser : choisissez l'Explorer. La nouvelle fenêtre qui s'ouvre alors (Weka Knowledge Explorer) présente les onglets suivants :
– Preprocess : pour choisir un fichier, inspecter et préparer les données.
– Classify : pour choisir, appliquer et tester différents algorithmes de classification : là, il s'agit d'algorithmes de classification supervisé.
– Cluster : pour choisir, appliquer et tester les algorithmes de clustering (apprentissage non supervisé).
– Associate : pour appliquer des algorithmes de génération de règles d'association.
– Select attributes : pour les algorithmes de filtrage d'attributs.
– Visualize : visualiser les données.
On utilisera certaines options en ligne de commande, pour cela il faut rendre la bibliothèque accessible avec :

```
export CLASSPATH=/chemin/vers/weka.jar
```

2.1 Prétraitement

A partir de l'onglet 'Preprocess', chargez le fichier de données 'vote.arff' dans le répertoire data de weka. Ces données décrivent l'appartenance politique de députés américains (démocrates ou républicains) en fonction de leurs votes ou positions sur différentes sujets. Vérifiez le nombre d'instances, d'attributs, explorez la répartition des classes par rapport aux attributs (fenêtre en bas à droite).
A ce stade on peut appliquer un certain nombre de filtres sur les données, pour normaliser certains attributs, en sélectionner certains, etc. Nous y reviendrons.

2.2 Visualisation

La visualisation de données peut permettre de se faire une idée de l'organisation de celles-ci. Cliquez sur l'onglet Visualize. Vous voyez un tableau de graphiques étiquettés par les attributs des données. Chaque graphique correspond à un graphique où chaque point représente une instance colorée en fonction de sa classe. La position du point est donnée en abscisse par l'attribut situé en haut du tableau et en ordonnée par l'attribut situé à gauche du tableau. Vous pouvez utiliser le réglage de "jitter" pour mieux voir les répartitions sur les attributs discrets. Quels sont les attributs qui semblent les plus utiles ?

2.3 Classification

La classification est l'apprentissage supervisé. Les algorithmes de classification prennent en entrée un ensemble de données étiquettées et renvoient des modèles qui permettent de classer de nouvelles données non étiquettées. Par défaut le classifieur choisi est ZeroR. Il s'agit du classifieur qui choisit la classe majoritaire. Lancez ce classifieur sur le jeu de données vote.arff. L'algorithme renvoie un résumé des résultats obtenus, puis le détail de l'efficacité de l'algorithme par classe et enfin la matrice de confusion.
Les classifieurs sont classés en 7 classes :

bayes – comporte notamment naïve Bayes et les réseaux Bayésiens ;

functions – comporte les réseaux de neurones, les SVM, les régressions linéaires... ;

lazy – comporte IB1 (le plus proche voisin) et IBk (les k plus proches voisins) ;

meta – comporte des algorithmes comme le Boosting notamment AdaBoost ;

misc – Divers algorithmes exotiques ;

trees – contient les algorithmes à base d'arbres de décision, dont le classique C4.5 listé sous le nom J48 ;

rules – des règles d'apprentissages comme le choix de la classe majoritaire ou apprendre en utilisant un unique attribut.

- Essayez les algorithmes que vous connaissez en explorant leurs paramètres. En particulier testez le comportement de J48 et de naïve Bayes.
- Expliquez et constatez les différentes options de tests, c'est-à-dire :
- utilisez l'ensemble d'apprentissage comme ensemble test ;
- utilisez un ensemble donné ;
- utilisez de la validation croisée ;
- utilisez une découpe de l'échantillon d'apprentissage.
- Expliquez les problèmes liés à l'utilisation de l'ensemble d'apprentissage pour l'ensemble qui permet d'effectuer les tests de qualité de l'apprentissage.

Lorsque des classifications sont effectuées, dans la zone Result list, en cliquant avec le bouton droit, une liste d'options apparaissent. En particulier il y a l'option "visualize classifier errors" qui permet d'enter en mode visualisation et qui encadre les instances erronées. Une autre façon de constater ce qui s'est produit est de cliquer sur le bouton More options.... Une fenêtre d'option s'ouvre dans laquelle on peut choisir l'option "Output predictions".

2.4 Choix d'attributs

Lorsque le nombre de données est trop grand ou que l'algorithme d'apprentissage à utiliser demande trop de ressources, il est souvent nécessaire de sacrifier de l'information pour gagner du temps de calcul. Se pose alors la question du choix des informations à éliminer. Allez dans Select attributes et choisissez comme évaluateur d'attribut CfsSubsetEval et comme méthode de recherche BestFirst. Cliquez sur start.

– Interprétez ce qu'il y a dans la zone Attribute selection output. Ce résultat correspond-t-il au choix que vous aviez fait lors de la visualisation des données ?
– Apprenez en utilisant seulement les attributs sélectionnés. La qualité de l'apprentissage est-elle meilleure ou moins bonne ?
– Comment la sélection d'attributs peut-elle améliorer la qualité de l'apprentissage ?

2.5 Utilisation de la ligne de commande

Toutes les opérations accessibles depuis l'interface peuvent être lancées en ligne de commande, ce qui permet plus de souplesse et économise la mémoire (ce qui est crucial sur des grands jeux de données).
On peut ainsi lancer un classifieur en spécifiant les données d'apprentissage :

```
java weka.classifiers.bayes.NaiveBayes -t vote.arff
```

L'évaluation se fait alors par validation croisée (10-fold). On peut spécifier un fichier de test avec l'option -T.
D'autres options utiles :
-d permet de sauver le modèle entraîné dans un fichier pour l'utiliser plus tard sur des données de test
-l permet à l'inverse de charger un modèle pour prédire sur de nouvelles données
-p permet d'afficher les prédictions sur les données de test, ainsi que certains attributs
-distribution : avec l'option -p, affiche aussi pour chaque instance les scores de chaque classe.
Ainsi :

```
java weka.classifiers.bayes.NaiveBayes -t train.arff -T test.arff -d NB.model
```

Entraine un classifieur Bayes naïf sur les données train.arff, teste sur les données test.arff, et sauve le modèle entraîné dans le fichier NB.model.
Ensuite :

```
java weka.classifiers.bayes.NaiveBayes -l NB.model -p 1,4,5 -distribution -T test.arff
```

Lance le modèle précédent sauvegardé sur les instances de test.arff, affiche les résultats pour chaque instance en affichant aussi les attributs 1, 4 et 5, ainsi que la probabilité attribuée à chaque classe par le modèle.
Vous pouvez tester ceci en utilisant vote.arff à la fois en entraînement et en test

3 Combiner apprentissage automatique et contraintes

Beaucoup de problèmes réels sur lesquels on veut faire de l'apprentissage automatique ne se limitent pas à un cadre de classification d'instances isolées (vous en avez déjà vu un exemple en reconnaissance de la parole).

Nous allons tester un cadre où l'apprentissage de classes doit respecter des contraintes formelles sur la formation des solutions. On peut considérer alors que la prédiction de la meilleure solution est un problème d'optimisation (avoir la meilleure confiance dans les prédictions) sous des contraintes diverses.

3.1 Application

L'exemple d'application est l'analyse superficielle de phrases ("text chunking"). Cela consiste à découper un texte en morceaux syntaxiques grossiers, ce qui permet une analyse partielle de ce qu'il contient.
Par exemple, la phrase "Le président de Total a admis un bénéfice record pour l'année précédente." peut être découpée en :

```
[Le président de Total] [a admis] [un bénéfice record] [pour] [l'année précédente].
```

Ce qui permet de trouver les acteurs et les objets dont on parle, ainsi que les actions et les circonstances. En fait on veut aussi définir des types d'unités reconnues : groupes nominaux, verbaux, prépositionnels et la phrase précédente serait analysée comme suit :

```
[NP Le président de Total] [VP a admis] [NP un bénéfice record] [PP pour]
[NP l'année précédente].
```

On prend ici les notations anglaises (NP = groupe nominal, VP= groupe verbal, PP = preposition) car les données que l'on utilisera sont en anglais.
Ce problème peut être vu comme un problème de classification de chaque mot, où l'on doit dire si un mot début un groupe, se trouve dans un groupe, ou bien est en dehors de tout groupe. On marquera par exemple le début d'un groupe nominal par B-NP (begin NP), un mot à l'intérieur par I-NP (inside NP), etc et O pour marquer un mot à l'extérieur de tous les groupes (outside).
Un exemple du corpus anglais :

```
He          B-NP
reckons     B-VP
the         B-NP
current     I-NP
account     I-NP
deficit     I-NP
will        B-VP
narrow      I-VP
to          B-PP
only        B-NP
1.8         I-NP
billion     I-NP
in          B-PP
September   B-NP
.           O
```

Pour prédire la bonne classe on utilise généralement les mots, leur catégorie syntaxique de base (nom, adjectif, etc) et le contexte immédiat (mot précédent, mot suivant, catégories de ceux-ci).
Vous trouverez sur moodle les jeux de données correspondants :
– small.arff est un petit jeu de données des mots isolés, qui permet d'explorer un peu le problème avec weka

– train.txt.arff est un gros ensemble pour entraîner un modèle plus réaliste, à traiter en ligne de commande
– test.txt.arff est un jeu de test pour évaluer le modèle précédent.

Ces données sont au format weka, les données d'origine sont lisibles dans les fichiers train.txt, test.txt.

3.2 Contraintes

Le problème du chunking va au-delà de la simple classification des mots séparément. D'une part ce que l'on veut retrouver est un ensemble de groupements de mots, et donc on veut trouver le début et la fin de chaque groupe pour avoir un découpage correct. Vous trouverer un script perl sur moodle (conlleval.pl) qui permet d'avoir l'évaluation de ces groupes (cf section A faire).

D'autre part, la séquence des étiquettes pour une phrase doit respecter un certain nombre de contraintes, par exemple :

– un mot à l'intérieur d'un groupe X est forcément dans une séquence introduite par un B-X. Dit autrement, un I-X est forcément précédé par un I-X ou un B-X, et il doit y avoir un B-X au début de la séquence.
– certains types groupes ne peuvent se suivre immédiatement sans être dans le même bloc : par exemple on ne peut avoir deux B-VP consécutifs ou deux B-PP consécutifs.

On peut vouloir contraindre plus globalement la structure des phrases aussi, par exemple en imposant :

– qu'il y ait au moins un groupe VP par phrase
– qu'il y ait toujours au moins un NP avant un VP
– etc.

Si l'on dispose des probabilités de chaque étiquette de mot grâce à un classifieur simple on peut alors considérer le problème d'optimisation suivant : trouver les étiquettes ayant la meilleure probabilité totale qui satisfont les contraintes sur une phrase.

3.3 A faire

En résumé, vous devez :

1. entraîner un classifieur sur les données d'entraînement train.txt.arff, et sortir ses prédictions sur les données de test.txt.arff, en gardant la distribution des scores pour chaque classe, en utilisant weka en ligne de commande.
2. pour une phrase donnée de test, générer le problème correspondant en ILP. Vous respecterez au moins la première contrainte introduite ci-dessus (pas de I-X isolé), et ajouterez les autres si le temps le permet. Vous pouvez tester avec les phrases 3 et 4 des données de test.
3. A partir de la solution générée par scip, générez le format attendu par le script d'évaluation. Celui-ci s'attend à avoir un mot par ligne avec la classe réelle et la classe prédite par le système, par exemple :

```
Rockwell B-NP B-NP
said B-VP B-VP
the B-NP B-NP
...
```

Chaque phrase est de plus séparée par une ligne vide, ajoutez donc une ligne vide à la fin du fichier. Il suffit ensuite de faire "perl connl2eval.pl < maphrase" pour avoir les statistiques.

4. Comparez à la solution sans contrainte que vous avez générée à l'étape 1.
5. Idéalement, on veut une estimation de cela sur tout le corpus de test ... il faut donc séparer chaque phrase, générer un fichier .lp pour chacune et lancer scip sur chaque problème... les script shell sont vos amis, par exemple vous pouvez faire quelque chose comme ceci (en supposant que toutes vos sorties sont dans un répertoire out) :

```
#!/bin/sh

SCIP=/chemin/vers/scip

for f in out/*.lp ;
do $SCIP -c \
"read $f  optimize write solution out/‘basename $f lp‘sol quit"\
 > log 2> logerr ;
done;
```

Et quelque chose du même genre pour retraduire toutes les solutions

Le but final est de battre le score sur le corpus de test réalisé le plus directement possible (pas de sélection de traits, classifieur bayes naif), à savoir une exactitude sur les étiquettes séparées de 92.71%, et des précisions/rappels sur les groupes de 86.21 et 89.35%, et en ajoutant simplement la première contrainte, le classifieur sous contraintes arrive à une exactitude de 92.91% et sur les groupes 87.19 et 89.35 de précision et rappel.

Nota Bene : l'exemple de tâche considérée ici est généralement traité IRL avec des algorithmes séquentiels proches de ceux que vous avez vu en reconnaissance de la parole, et des traits plus complexes, avec des résultats légèrement supérieurs, cf `http://www.cnts.ua.ac.be/conll2000/chunking/`.