AFD 2013-2014
Enseignant : Vincent Guigue

Compte-rendu

Classification de textes avec Weka

AYRES Romain

Date de remise : 09/01/2014

Table of Contents

Introduction

L'objectif du projet est la prise en main du logiciel Weka et de ses fonctionnalités. Les deux travaux portent sur la classification de textes. Le premier consiste à prédire l'orateur d'une phrase («authoring») tandis que le second porte sur une analyse de sentiments concernant des avis de consommateurs (l'avis est-il positif ou négatif ?).

Le rapport est ainsi divisé en deux parties (pour les deux travaux), présentant chacune une analyse des modèles étudiés (pre-processing, algorithmes retenus, post-processing) et des résultats obtenus.

Chirac/Mitterrand : qui est l'orateur ?

Introduction

Objectif

L'objectif de la tâche est de construire un modèle pouvant prédire si une phrase a été prononcée par Jacques Chirac ou François Mitterrand. Ce n'est donc pas de l'authoring «classique» dans le sens où les phrases étudiées ne sont pas écrites par Chirac et Mitterrand, mais seulement prononcées. C'est un détail important lors du choix des options de pre-processing sur lesquelles nous reviendrons plus tard.

Données utilisées

La base d'apprentissage contient environ 57000 phrases prononcées étiquetées (C ou M) et la base de test en contient environ 27000 (non étiquetées).

Segment du fichier d'apprentissage
C "A Brazzaville, que fut scellée la première union régionale des pays africains francophones."
C "A Brazzaville, que l'Afrique de demain se dessine."
M "Je ne sais ni pourquoi ni comment on s'est opposé il y a quelques douze années..."

Segment du fichier de test
Y "En répondant à votre invitation, en effectuant cette première visite ..."
Y "Et ce moment exceptionnel, j'ai essayé de le concrétiser..."
Y "Et le musée."

Métrique d'évaluation

Les classes sont déséquilibrées, il y a beaucoup plus de phrases prononcées par Chirac que de phrases prononcées par Mitterrand. Ainsi, l'évaluation utilisée n'est pas le taux de bonne classification qui ne refléterait pas correctement la performance de l'algorithme mais le score F1 de la classe M qui est une moyenne pondérée de la précision et du rappel et qui reflète beaucoup mieux la qualité d'un algorithme d'apprentissage dans le cas de classes déséquilibrées. Voici la formule du score F1 utilisé :

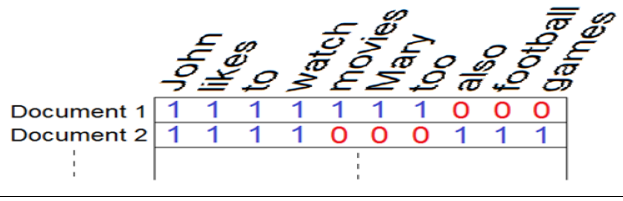
$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Pre-proccessing

Pour faire de la classification de textes, il faut déjà pouvoir représenter les textes sous une forme manipulable par la machine. Ainsi, il faut projeter chaque texte étudié sur un dictionnaire de taille choisie. Chaque texte sera donc représenté par un vecteur de la taille du dictionnaire contenant des 0

(si le mot du dictionnaire n'est pas dans le texte) et des 1 (ou une autre valeur en fonction du codage utilisé si le mot du dictionnaire figure dans le texte). C'est la représentation en sac de mots («bag of words»).

Voici un exemple de représentation en sac de mots (codage présentiel) :

Textes	Sacs de mots correspondants
« John likes to watch movies. Mary likes too. » « John also likes to watch football games. »	

Cette conversion en représentation sac de mots n'est pas anodine: il y a de nombreux paramètres à considérer en fonction de la tâche souhaitée. Ici, on souhaite déterminer l'orateur d'un texte. Il serait fastidieux et trop long de tester l'ensemble des paramètres possibles. Il faut donc avant toute chose se poser la question pour chaque paramètre: que cherche-t-on à faire? sur quelles caractéristiques nous attarderions nous si c'était à nous d'annoter chacune de ces phrases? quels détails regarderions nous si c'était à nous de prédire l'orateur? Passons sur l'ensemble des options et essayons de déterminer « à priori » la valeur appropriée pour notre tâche.

Codage (TF-IDF, présentiel, fréquentiel)

Pour cette tâche, le TF-IDF ne semble pas être le plus approprié car il vise à donner un poids moins important aux termes les plus fréquents, considérés comme moins discriminants. Cependant, ce n'est pas une bonne idée ici car les mots fréquents utilisés de différentes manières par les orateurs peuvent permettre une bonne discrimination.

Le codage présentiel (0 ou 1) est à première vue (et pour la même raison) inintéressant, dans le sens où on perd l'information de la fréquence d'un mot, information qui a son importance lorsque l'on veut faire de l'authoring (à l'inverse de l'analyse de sentiments où la fréquence d'un mot a tendance à détériorer les performances de l'algorithme).

Pour notre tâche, le codage fréquentiel semble être le plus approprié, il conserve l'information de fréquence d'utilisation de tous les mots, même les plus fréquents.

Conversion en minuscules (true/false)

Comme il s'agit d'un texte prononcé et non écrit, il serait logique d'effectuer la conversion en minuscules. Les lettres majuscules dans les phrases considérées correspondent aux noms propres et aux premiers mots de chaque phrase. Les deux présidents ont des techniques d'élocution différentes et commencent certainement leurs phrases différemment. Ainsi, effectuer la conversion en minuscule retire finalement de l'information qui pourrait être intéressante.

Stemmatisation (true/false)

Il peut être intéressant d'ajouter de la lemmatisation (réduction des mots à leurs racines) afin de réduire la dimension du problème et par conséquent de réduire le risque de sur-apprentissage. Il est difficile d'avoir un avis à priori sur les conséquences de la lemmatisation sur la performance de notre algorithme. Nous testerons donc ultérieurement l'influence de ce paramètre.

Utilisation d'une stoplist (true/false)

Le retrait de «stop words» permet de réduire la dimension de l'espace et de retirer les mots non discriminants (par exemple, les mots "donc", "pour" et "alors" sont très utilisés mais n'ont à priori aucun pouvoir discriminant). La stoplist utilisée est la suivante:
http://www.ranks.nl/stopwords/french.html

Tokenization

La «tokenization» consiste à découper le texte afin de créer le dictionnaire. Différents sous-paramètres sont à considérer: les délimiteurs et les n-grammes.

Les délimiteurs utilisés sont les délimiteurs classiques : \r\n\t.,;:"()?! (espaces, tabulations, sauts de lignes,..).
Concernant les n-grammes, il peut être intéressant de ne pas se limiter aux mots isolés, qui à cause de la représentation sac de mots, sont totalement indépendants des mots précédents et suivants. Ainsi, on peut capturer un peu mieux la structure de la phrase en considérant les bi-grammes (groupes de deux mots consécutifs) en plus des uni-grammes. La prise en compte est tri-grammes est rarement utilisée car même si elle apporte encore davantage d'information sur la structure de la phrase, elle apporte aussi une complexité de modèle trop importante et ainsi un risque de sur-apprentissage. Le tokenizer utilisé et permettant de capturer les n-grams est « NGramTokenizer ».

Nombre de mots à conserver

Ce paramètre définie la taille du dictionnaire souhaité. Conserver 1000 mots par classe peut sembler trop faible (risque de sous-apprentissage) tandis qu'en garder 100 000 par classe peut sembler trop élevé (risque de sur-apprentissage). Un bon compromis est d'utiliser entre 20 000 et 25 000 mots par classe (contenant uni-grammes et bi-grammes).

Apprentissage

Algorithme d'apprentissage utilisé

Plusieurs algorithmes ont été testés (Decision trees, Perceptron, WLSVM, NaiveBayes, NaiveBayesMultinomial). Cependant, en raison de la quantité de données d'apprentissage (57000) et de la dimension associée (25000*2), le seul algorithme fonctionnant assez rapidement pour effectuer une multitude de tests est le NaiveBayesMultinomial (classifieur bayésien naïf avec une distribution polynomiale pour chaque mot).

En effet, la simplicité de cet algorithme réside dans la supposition que tous les mots considérés sont indépendants. Ainsi, il calcule pour chaque mot une probabilité à posteriori d'appartenir à la classe Chirac ou à la classe Mitterrand. Enfin, la prédiction s'effectue en choisissant la classe maximisant la probabilité à posteriori que la phrase appartienne à cette classe (cette probabilité étant obtenue par une multiplication de chacun des mots composant la phrase).

Evaluation

Comme précisé dans l'introduction, la métrique d'évaluation utilisée ici est le score F1 car les classes sont déséquilibrées. Le score F1 est bien entendu calculé via cross-validation, afin d'avoir une bonne estimation du score qu'on obtiendrait sur des nouvelles données.

Post-processing

Structure des fichiers d'apprentissage et de test

On pourrait se contenter des performances obtenues à ce stade mais ce serait passer à côté d'une astuce propre à la compétition. En effet, en observant les données d'apprentissage, on remarque que celles-ci sont disposées d'une manière bien particulière, elles sont disposées par bloc. Ainsi, on a un bloc de phrases prononcées par Chirac, suivi par un bloc de phrases prononcées par Mitterrand, puis suivi par un bloc de phrases prononcées par Chirac, etc. En supposant que le fichier de test est conçu d'une manière similaire, il peut être très intéressant au point de vue performance de prendre en compte ce format.

Solution proposée

La solution proposée est un lissage du fichier de prédictions obtenu suite à notre apprentissage. Il s'agit en effet d'essayer de reconstituer les blocs présents dans le fichier de test. Pour cela, un parcours des prédictions est effectué, et pour chaque prédiction, on regarde ses 10 voisins les plus proches et on effectue une moyenne pondérée sur la distribution calculée en accordant un poids plus important aux

prédictions Mitterrand (2,85 contre 1 pour Chirac) car elles sont beaucoup moins présentes. On peut donc observer ce type de transformation :

Prédictions initiales	Prédictions lissées
C	C
C	C
C	C
C	C
M	M
C	M
C	M
M	M
M	M
C	M
C	C
C	M
C	C
C	C
C	C
C	C
M	C
C	C

On obtient bien des blocs qui pourraient sans doute correspondre aux blocs initiaux. Cependant, cette technique ne suffit pas car il peut toujours y avoir des incohérences de type (MMMCM ou CCMCC). Un « sous-lissage » est alors effectué pour supprimer ces anomalies :

Prédictions lissées	Prédictions finales
C	C
C	C
C	C
C	C
M	M
M	M
M	M
M	M
M	M
M	M
C	M
M	M
C	C
C	C
C	C
C	C
C	C
C	C

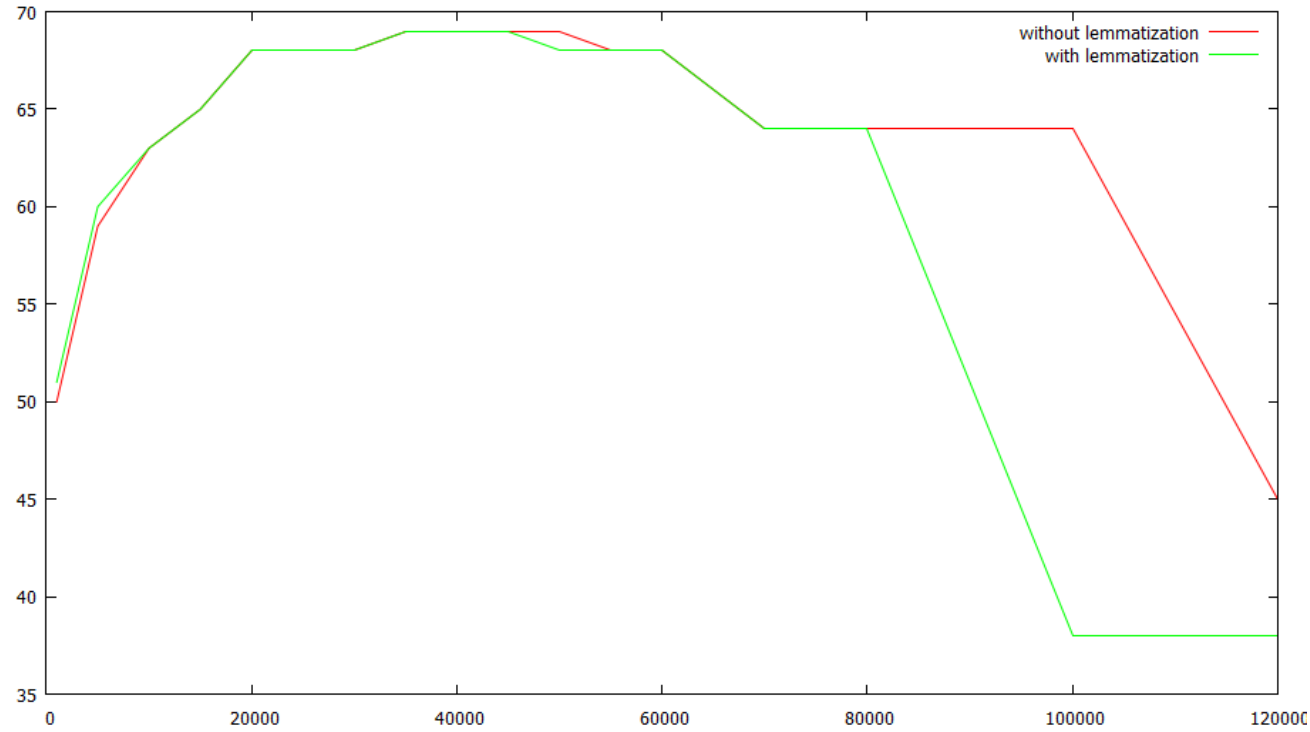
La conséquence de ce lissage sur le score F1 est une augmentation d'environ 15%.

On pourrait trouver davantage d'améliorations à cette étape de post-processing mais ce n'est pas l'objectif principal ici.

Expériences réalisées

Influence de la taille du dictionnaire et de la stemmatisation

On s'intéresse ici à l'influence des paramètres de pre-processing sur la performance de notre modèle. Ainsi, on va faire varier le nombre de mots conservés lors de la segmentation. Nous le faisons varier de 1000 à 100 000, par pas de 5000 et nous observons les performances obtenues. Les scores F1 sont calculés par validation croisée (5 séparations) avant post-processing :



Scores F1 obtenus en validation croisée en fonction du nombre de mots considérés par classe

Les courbes obtenues n'apportent pas d'information pertinente sur la nécessité d'utiliser la lemmatisation ou non. En effet, les performances sont à peu près identiques.

En revanche, ces courbes nous donnent une indication sur le nombre de mots à considérer par classe pour obtenir un bon modèle. Une valeur entre 20000 et 60000 semble faire l'affaire et renvoie les meilleurs résultats. Ce chiffre prend en compte les uni-grammes mais aussi les bi-grammes.

Etude du dictionnaire et des probabilités associées

En observant les probabilités calculées pour chaque mot, on peut s'intéresser à quelques mots les plus discriminants pour la tâche:

x	*p(M|x)*	*p(C|x)*
bien	0.002282660734364692	0.0014131478702238299
de	0.010287332589674479	0.02016695635549257
et de	0.0011246285933224623	0.002818518392391405
monsieur	2.4086863563035327E-4	2.0203051434847032E-6
un	0.00499739181910282	0.00651552913539147

La première observation est que l'on retrouve des mots simples (« bien », « de », « un ») que l'on pouvait imaginer comme étant des stop words. En réalité il ne sont pas présents dans la stoplist, est-ce un oubli de la part de l'auteur ou un choix volontaire?
Ensuite, on remarque que la probabilité à posteriori d'être dans la classe Mitterrand quand on rencontre les mots « bien » et « monsieur » est beaucoup plus élevée que la probabilité à posteriori d'être dans la classe Chirac. En revanche pour les mots « de », « et de » et « un », on a plus de chance d'être dans la classe Chirac. Ces mots sont donc des bons indicateurs pour discriminer les deux orateurs (mais il y en a beaucoup d'autres).

Avis de consommateurs : analyse de sentiments

Introduction

Objectif

L'objectif de la tâche est de construire un modèle pouvant prédire si le commentaire d'un internaute est plutôt positif ou plutôt négatif (classification binaire).

Données utilisées

La base d'apprentissage contient 2000 revues étiquetées (1000 négatives, 1000 positives) et la base de test en contient 25000.

Métrique d'évaluation

Le problème est plus simple que le précédent car les classes sont équilibrées. On peut donc utiliser le taux de bonne classification comme indicateur de performance de notre algorithme.

Pre-proccessing

Comme pour la première tâche, il faut déjà déterminer quelles options pourraient nous intéresser lors de la création du sac de mots. Passons sur l'ensemble des options et essayons de déterminer « à priori » la valeur appropriée pour notre tâche.

Codage (TF-IDF, présentiel, fréquentiel)

Il est difficile d'avoir un à priori sur un bon codage pour la tâche étudiée. Le TF-IDF pourrait être intéressant pour accorder un poids plus important aux mots les plus discriminants, tandis que le codage présentiel reste simple et efficace. Une comparaison qualitative des différents codage est effectuée en fin de rapport.

Conversion en minuscules (true/false)

Bien que la conversion en minuscules permet de réduire la taille du dictionnaire et par conséquent de simplifier le problème, les majuscules peuvent être un très bon indicateur de sentiment (« BEST MOVIE EVER! ») et seront conservées.

Stemmatisation (true/false)

Ici encore, le choix de la stemmatisation est difficile à prendre à priori. Nous testerons donc ultérieurement l'influence de ce paramètre.

Utilisation d'une stoplist (true/false)

Le retrait de « stop words » permet de réduire la dimension de l'espace et de considérer seulement des mots discriminants (par exemple, les mots "donc", "pour" et "alors" sont très utilisés mais n'ont à priori aucun pouvoir discriminant). La stoplist utilisée est la liste de base fournie par Weka.

Tokenization

Les délimiteurs utilisés sont les délimiteurs classiques (\r\n\t.,;:"()*-=&) présentés précédemment auxquels on a retiré le point d'exclamation et le point d'interrogation qui sont des marqueurs de sentiments qu'il faut conserver (et auxquels on a ajouté *-=&).
Comme précédemment, nous utilisons le Tokenizer « NGramTokenizer » qui permet de prendre en compte les bi-grammes en plus des uni-grammes.

Nombre de mots à conserver

Nous conservons pour cette tâche environ 15000 mots par classe.

Apprentissage

Algorithme d'apprentissage utilisé

Plusieurs algorithmes ont été testés (Perceptron, WLSVM, NaiveBayes, NaiveBayesMultinomial). Comme pour la première tâche, le classifieur le plus rapide est le NaiveBayesMultinomial, c'est donc avec ce dernier que la majorité des expériences ont été réalisées.

Une expérience comparative entre WL-SVM et NaiveBayesMultinomial est présentée en fin de rapport.

Post-processing

Pour cette tâche, l'étape de post-processing est absente car nous n'avons à priori aucune information sur la manière dont à été fabriqué le fichier de test (probablement aléatoirement).

Expériences réalisées

Influences du codage, de la stemmatisation et de l'algorithme utilisé

On s'intéresse ici à l'influence du codage utilisé, de la stemmatisation et de l'algorithme d'apprentissage utilisé. Voici le tableau récapitulatif des résultats :

Codage	Stemmatisation	WL-SVM (linear kernel) CV 3-fold	NaiveBayesMultinomial CV 3-fold
TF IDF	NON	90,15%	89,15%
TF IDF	OUI	89,25%	89,35%
TF	NON	86,90%	87,60%
TF	OUI	87,25%	87,80%
Présence	NON	86,90%	88,00%
Présence	OUI	87,25%	88,05%

Si les performances semblent anormalement élevées, c'est parce que les résultats sont biaisés. En effet, l'étape de pre-processing est effectuée avant la validation croisée, toutes les phrases de la base d'apprentissage sont donc projetées sur le dictionnaire calculé à partir de celles-ci. Lors de l'utilisation de la validation croisée, on sépare les données d'apprentissage en trois parties, l'une devant être considérée comme une base de test. Cependant, cette partie a servi lors du calcul initial du dictionnaire et de la représentation en sac de mots, ce ne sont donc pas des données complètement nouvelles, et le résultat se retrouve biaisé. Il faudrait en fait recalculer un nouveau dictionnaire à chaque étape de validation croisée, mais le framework Weka ne permet pas de le faire simplement. Les résultats obtenus sur les vraies données de test sont d'environ 8% moins élevés.

Il reste néanmoins possible de comparer les options utilisées. On remarque que le codage présentiel est préférable au codage fréquentiel, ce qui n'était pas forcément attendu à priori. Le codage TF-IDF donne de meilleurs résultats que les deux précédents, mais de moins bons sur les réelles données de test, ce qui s'explique par un biais encore plus important lors de la création du dictionnaire (on affecte un poids à un mot en fonction du nombre d'apparition de ce mot dans l'ensemble de la classe, ce qui revient donc à apprendre sur des données de validation).

La stemmatisation a tendance à augmenter légèrement les performances dans la plupart des cas. Enfin, même si les SVM ont tendance à donner des meilleurs résultats que NaiveBayesMultinomial avec un codage TF-IDF, ce n'est pas le cas avec un codage présentiel.

Le meilleur score obtenu sur les vraies données de test est obtenu avec les options suivantes :
Codage présentiel

- Stemmatisation OFF
- UseStopList ON
- LowerCaseTokens OFF
- WordsToKeepPerClass = 16000

- Tokenizer utilisé : NGramTokenizer avec uni-grammes + bi-grammes
- Délimiteurs : \r\n\t.,;:\"()*-=&
- Classifieur NaiveBayesMultinomial

Visualisation des mots les plus discriminants

En utilisant l'algorithme SMO fourni par Weka, il est possible de visualiser les mots considérés comme les plus négatifs et les mots considérés comme les plus positifs :

Most negative words	Most positive words
-0.1193 waste	0.0996 hilarious
-0.1133 poor	0.0951 memorable
-0.1102 bad	0.0846 fun
-0.109 plot	0.0809 by a
-0.1087 worst	0.0795 excellent
-0.1087 boring	0.0791 very well
-0.1052 the only	0.0779 terrific
-0.1036 should have	0.0757 performances
-0.0981 awful	0.0745 entertaining
-0.0876 mess	0.0715 perfectly
-0.078 to waste	0.0698 he is
-0.0778 script	0.0687 simple
-0.0761 falls	0.0687 back
-0.0753 on it	0.0682 people
-0.0713 ridiculous	0.0657 one of
-0.0709 flat	0.0641 class
-0.0706 would have	0.064 change
-0.0687 lame	0.0633 well the
-0.0676 supposed	0.0627 though it
-0.0669 reason	0.0626 great
-0.0659 stupid	0.0609 due to
-0.0657 wasted	0.0597 many of
-0.0654 to work	0.0592 masterpiece
-0.0654 the worst	0.0592 flaws
-0.0642 potential	0.0591 and it's
-0.0641 horrendous	0.059 pace
-0.0634 style	0.059 i've
-0.0633 to show	0.0586 wonderfully
-0.0632 video	0.0584 days
-0.0626 tv	0.0581 solid
-0.061 i'm not	0.058 men
-0.0609 supposed to	0.058 leave
-0.0597 could have	0.0578 the best
-0.0595 attempt	0.0572 enjoyable
-0.0591 to be	0.0566 visually
-0.059 annoying	0.0561 outstanding

Ces résultats sont très intéressants, ils permettent d'observer les mots les plus discriminants pour la tâche. Ainsi, « waste » est le mot le plus négatif et « hilarious » est le plus positif. On remarque que contrairement à ce que l'on pouvait s'attendre, le mot « good » n'apparait pas dans les mots positifs, car il apparait certainement aussi dans les commentaires négatifs (« not good »). De plus, on peut souligner l'importance d'avoir considérer les bi-grammes ici : « very well », « the best », « should have », « the worst »...

Negative words

Positive words

Conclusion

La classification automatique de textes, quel que soit l'objectif fixé (authoring, analyse de sentiments, segmentation de sujets, compréhension,..) est un problème difficile car nous traitons des données non numériques. Or les algorithmes utilisés fonctionnent avec des données numériques, il faut donc préalablement transformer les textes sous une forme numérique (pre-processing) et cette étape fait perdre énormément d'informations (par exemple, la création d'un sac de mots supprime l'information de localisation d'un mot dans une phrase).

Il est important lors du choix des options de l'étape de pre-processing de se poser la question : que regarderions nous si c'était à nous de réaliser la tâche de classification ? Les réponses à cette question nous guident vers les bons choix d'options de pre-processing, étape la plus cruciale dans la chaîne de traitements utilisée pour la classification automatique de textes.