Techniques et applications du traitement automatique de la langue (TALN)
IFT-7022

TP2 – Classification et prétraitement de textes

Travail remis à :
Luc Lamontagne

Par:
Guillaume Chevalier
guillaume.chevalier.2@ulaval.ca
111 131 607
GUCHE29

Département d'informatique et de génie logiciel
Université Laval
13 Novembre 2018

TP2 - Expressions régulières et modèles n-grammes

Table des matières

Introduction

Ce travail fut réalisé seul.

Dans ce travail pratique, il est premièrement demandé de faire de la classification de sentiments sur des textes (sentiment positif et négatif) en utilisant plusieurs pipelines variés. Il est deuxièmement demandé de construire un pipeline de classification de la langue par un n-gram de caractères, pour classifier parmi les langues avec les labels suivants:

```
[(0, 'english'), (1, 'espanol'), (2, 'french'), (3, 'portuguese')]
```

Pour rouler le code, il est préférable d'utiliser les iPython notebooks (commande **"jupyter-notebook"**), lesquels furent exportés vers le **README.md** et aussi en fichiers **".py"** dans la racine du projet. Ma version de Python est Python 3.6, sous linux. Le code fut aussi exporté en fichiers ".py" automatiquement à partir des notebooks, ces fichiers se trouvent à la racine du projet.

Il y a aussi un répertoire important nommé **"src/"** dans lequel le code des pipelines réside.

Tâche 1 - Analyse de sentiment

Voici les éléments du pipeline. Un "ou" logique avec le symbole "|" signifie un choix à faire. Toutes les combinaisons possibles de choix ont été comparées. Et même, pour chacune de ces combinaisons, un grid search cross validation 5-fold a été fait pour optimizer les autres hyperparamètres. En d'autres mots, pour chaque combinaison "ou" logiques du tableau ci-dessous, un grid search est fait pour optimizer des paramètres supplémentaires du pipeline.

Étape 1.
Tokenizer avec NLTK

Étape 2.
Ne pas convertir en minuscule | Convertir en minuscule

Étape 3.
Conserver tous les attributs | Ne conserver que les mots de classe ouverte | Enlever les mots-outils

Étape 4
Ne pas ajouter d'attributs supplémentaires | Ajouter le nombre de mots positifs/négatifs

Étape 5.
Pas de stemming | faire du stemming avec le Porter Stemmer

Étape 6.
Count Vectorizer

Étape 7.
Logistic Regression (Classifier) | Multinomial Naive Bayes

Chaque élément (ou du moins, la majorité de ces éléments) des étapes est implémenté comme une classe qui hérite des classes BaseEstimator et TransformerMixin de Scikit-Learn pour faire un Pipeline scikit-learn. Voici une description pour chaque classe:

- **Tokenizer avec NLTK** : le word_tokenize de NLTK est wrappé pour le pipeline scikit-learn.
- **Convertir en minuscule** : pour chaque token, on converti les lettres en minuscule.
- **Conserver tous les attributs** : ne rien faire (on passe à la prochaine étape).
- **Ne conserver que les mots de classe ouverte** : le Part Of Speech (POS) tagger de scikit-learn est utilisé.
- **Enlever les mots-outils** : Les stop-words de scikit-learn ET de NLTK sont combinés pour être plus exclusif.
- **Ne pas ajouter d'attributs supplémentaires** : ne rien faire (on passe à la prochaine étape).
- **Ajouter le nombre de mots positifs/négatifs** : avec SentiWordNet, on prepend deux nouveaux tokens spéciaux pour les sentiments positifs et négatifs, ces tokens seront comptés dans la vectorization plus tard.
- **Pas de stemming** : ne rien faire (on passe à la prochaine étape).
- **Faire du stemming avec le Porter Stemmer** : le PorterStemmer est wrappé dans une classe de pipeline
- **Count Vectorizer** : le CountVectorizer de scikit-learn, mais avec des paramètres de façon à ne pas tokenizer, mettre en minuscules, ni enlever les mots-outils ni quoique ce soit, car ces étapes sont programmées et choisies manuellement dans les étapes précédentes.
- **Logistic Regression (Classifier)** : une régression logistique de scikit-learn.
- **Multinomial Naive Bayes** : MultinomialNB de scikit-learn.

La majorité de ces objets sont trouvables à cet emplacement dans le code : "src/pipeline_steps/*.py".

Résultats: les résultats sont calculés en ré-entraînant le meilleur classificateur par catégorie sur toutes les données d'entraînement et les tester sur les données de test. C.-à.-d., en premier, les données sont séparées en données d'entraînement et de test avec un split 80% et 20%.
De plus, la classification est sur les documents tests de longueur complète plutôt que séparés par phrase. Seul le corpus d'entraînement fut séparé par phrase.

Test set score for 'Logistic Classifier with_lowercase all_attributes with_pos_neg_attribute with_stemming ': 84.25%
Test set score for 'Logistic Classifier with_lowercase all_attributes with_pos_neg_attribute ': 81.5%
Test set score for 'Logistic Classifier with_lowercase all_attributes with_stemming ': 81.75%
Test set score for 'Logistic Classifier with_lowercase all_attributes ': 81.25%
Test set score for 'Logistic Classifier with_lowercase remove_stop_words with_pos_neg_attribute with_stemming ': 80.0%
Test set score for 'Logistic Classifier with_lowercase remove_stop_words with_pos_neg_attribute ': 81.5%
Test set score for 'Logistic Classifier with_lowercase remove_stop_words with_stemming ': 78.75%
Test set score for 'Logistic Classifier with_lowercase remove_stop_words ': 79.25%
Test set score for 'Logistic Classifier with_lowercase keep_open_classes_only with_pos_neg_attribute with_stemming ': 81.5%
Test set score for 'Logistic Classifier with_lowercase keep_open_classes_only with_pos_neg_attribute ': 81.25%
Test set score for 'Logistic Classifier with_lowercase keep_open_classes_only with_stemming ': 81.5%
Test set score for 'Logistic Classifier with_lowercase keep_open_classes_only ': 82.75%
Test set score for 'Logistic Classifier all_attributes with_pos_neg_attribute with_stemming ': 83.75%
Test set score for 'Logistic Classifier all_attributes with_pos_neg_attribute ': 81.75%
Test set score for 'Logistic Classifier all_attributes with_stemming ': 81.5%
Test set score for 'Logistic Classifier all_attributes ': 80.0%
Test set score for 'Logistic Classifier remove_stop_words with_pos_neg_attribute with_stemming ': 81.75%
Test set score for 'Logistic Classifier remove_stop_words with_pos_neg_attribute ': 80.75%
Test set score for 'Logistic Classifier remove_stop_words with_stemming ': 79.5%
Test set score for 'Logistic Classifier remove_stop_words ': 79.0%
Test set score for 'Logistic Classifier keep_open_classes_only with_pos_neg_attribute with_stemming ': 80.0%
Test set score for 'Logistic Classifier keep_open_classes_only with_pos_neg_attribute ': 81.25%
Test set score for 'Logistic Classifier keep_open_classes_only with_stemming ': 81.75%
Test set score for 'Logistic Classifier keep_open_classes_only ': 80.5%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase all_attributes with_pos_neg_attribute with_stemming ': 80.75%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase all_attributes with_pos_neg_attribute ': 79.5%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase all_attributes with_stemming ': 80.75%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase all_attributes ': 80.0%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase remove_stop_words with_pos_neg_attribute with_stemming ': 81.75%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase remove_stop_words with_pos_neg_attribute ': 79.25%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase remove_stop_words with_stemming ': 78.25%

Test set score for 'Multinomial Naive Bayes Classifier with_lowercase remove_stop_words ': 78.5%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase keep_open_classes_only with_pos_neg_attribute with_stemming ': 79.75%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase keep_open_classes_only with_pos_neg_attribute ': 79.25%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase keep_open_classes_only with_stemming ': 78.25%
Test set score for 'Multinomial Naive Bayes Classifier with_lowercase keep_open_classes_only ': 79.0%
Test set score for 'Multinomial Naive Bayes Classifier all_attributes with_pos_neg_attribute with_stemming ': 80.25%
Test set score for 'Multinomial Naive Bayes Classifier all_attributes with_pos_neg_attribute ': 79.75%
Test set score for 'Multinomial Naive Bayes Classifier all_attributes with_stemming ': 79.75%
Test set score for 'Multinomial Naive Bayes Classifier all_attributes ': 78.0%
Test set score for 'Multinomial Naive Bayes Classifier remove_stop_words with_pos_neg_attribute with_stemming ': 81.75%
Test set score for 'Multinomial Naive Bayes Classifier remove_stop_words with_pos_neg_attribute ': 78.5%
Test set score for 'Multinomial Naive Bayes Classifier remove_stop_words with_stemming ': 78.25%
Test set score for 'Multinomial Naive Bayes Classifier remove_stop_words ': 78.75%
Test set score for 'Multinomial Naive Bayes Classifier keep_open_classes_only with_pos_neg_attribute with_stemming ': 81.5%
Test set score for 'Multinomial Naive Bayes Classifier keep_open_classes_only with_pos_neg_attribute ': 80.0%
Test set score for 'Multinomial Naive Bayes Classifier keep_open_classes_only with_stemming ': 80.5%
Test set score for 'Multinomial Naive Bayes Classifier keep_open_classes_only ': 79.25%

Résultat. Le meilleur score est le suivant:
'Logistic Classifier with_lowercase all_attributes with_pos_neg_attribute with_stemming': 84.25%

Tâche 2 - Identification de la langue d'un texte

Pour cette partie, six modèles sont entraînés, soit une combinaison des classificateurs logistiques et par naive bayes, et pour chacun, l'utilisation de Bag of Words (BoW) de 1 à 3 caractères.

Voici les étapes du pipeline:

Étape 0. (faite en dehors du pipeline au préalable, seulement sur les données d'entraînement)
Convertir les documents en phrases avec la tokenisation de phrase de NLTK

Étape 1.
Convertir en minuscule

Étape 2.
Convertir en n-gram de caractères

Étape 3.
CountVectorizer de scikit-learn

Étape 4.
TfidfTransformer de scikit-learn

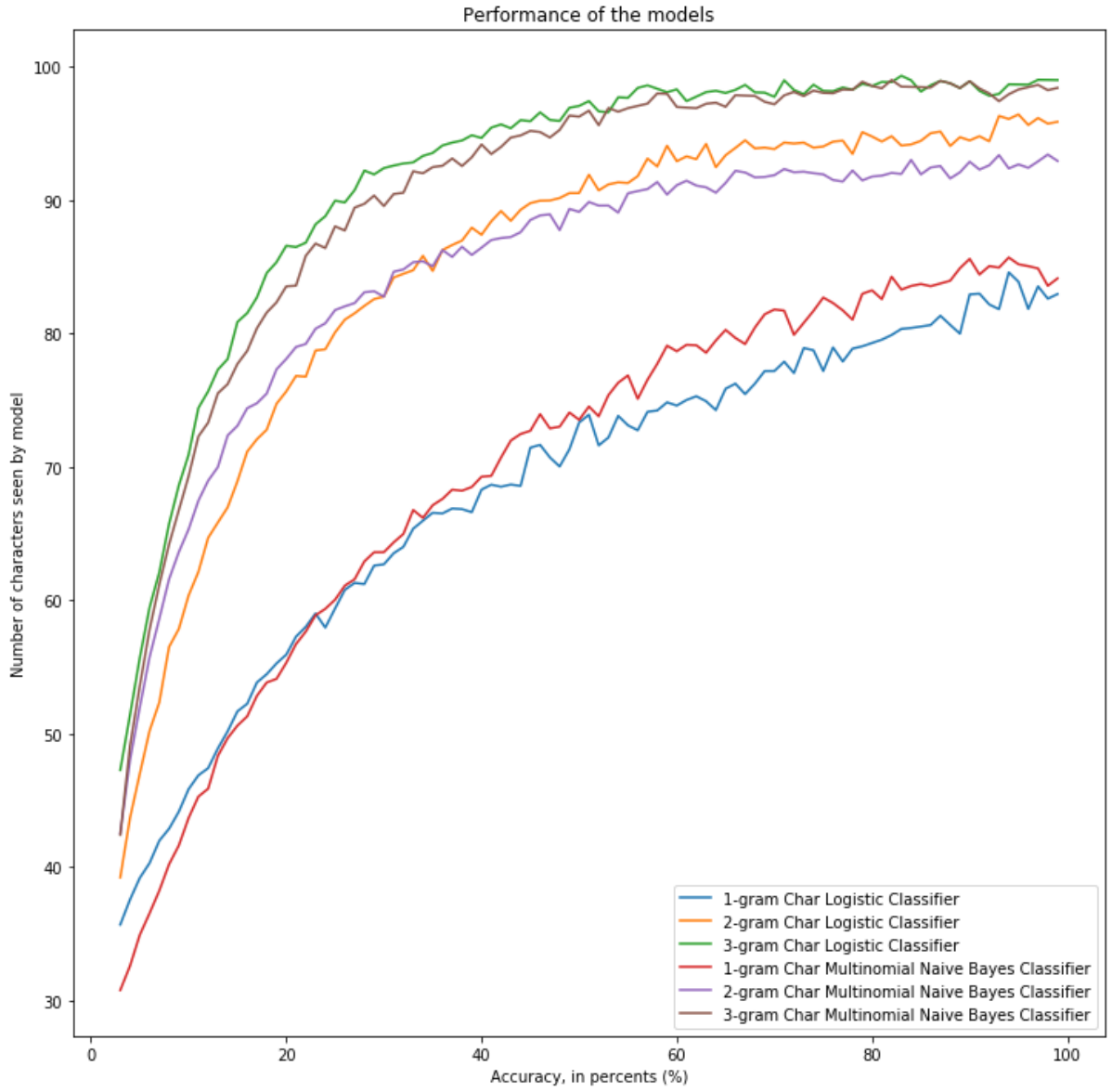
Étape 5.
Logistic Regression (Classifier) | Multinomial Naive Bayes

Comme pour la tâche 1, chaque élément (ou du moins, la majorité de ces éléments) des étapes est implémenté comme une classe qui hérite des classes BaseEstimator et TransformerMixin de Scikit-Learn pour faire un Pipeline scikit-learn, et un grid search cross validation 5-fold séparé et unique est fait pour chaque combinaison possible du pipeline. Voici une description pour chaque classe qui n'étaient pas déjà décrite dans la partie sur la tâche 1 dans le rapport:

- **Convertir en n-gram de caractères** : les documents sont convertis, par exemple:
  ["test!", "hey"] → [["tes", "est", "st!"], ["hey"]]
  Cela est fait en utilisant simplement ma librairie conv laquelle permet de faire des for-loop de convolution sur les lettres du texte, et en l'ayant wrappée dans un élément de pipeline scikit-learn: https://github.com/guillaume-chevalier/python-conv-lib
- **CountVectorizer** : Contrairement au CountVectorizer de la tâche 1, ici nous nous limitons en plus au unigrams, comme la séparation en n-gram est faite dans la classe précédente du pipeline présent.
- **TfidfTransformer de scikit-learn** : Le TfidfTransformer de scikit-learn avec paramètres par défaut.

Il est possible de voir les résultats à la **Figure 1**. Malgré que je suis seul et non en équipe de deux ou trois, j'ai décidé de générer la **Figure 1** quand même par curiosité, en bonus au travail demandé.

Figure 1 – Performance des six modèles n-gram sur le jeux de test, une fois ré-entraînés sur le jeux de test complet suite à une cross-validation de d'autres paramètres de leur pipeline.



Voici aussi les résultats lorsque les documents de test sont pris dans leur longueur intégrale pour les prédictions :

```
Score for '1-gram Char Logistic Classifier': 100.0%
Score for '2-gram Char Logistic Classifier': 100.0%
Score for '3-gram Char Logistic Classifier': 100.0%
Score for '1-gram Char Multinomial Naive Bayes Classifier': 95.0%
Score for '2-gram Char Multinomial Naive Bayes Classifier': 100.0%
Score for '3-gram Char Multinomial Naive Bayes Classifier': 100.0%
```

Conclusion

Pour conclure, l'identification de la langue se révèle triviale, alors que l'analyse de sentiments pourrait être améliorée.

Pour ce qui en est de l'analyse de sentiments, les résultats sont de **84.25%**. Ce score provient de l'évaluation sur le test set (20% des données 'held out'). Le choix du meilleur classificateur fut fait suite à des validation croisées sur les données d'entraînement (80% des données pré-séparées suite à un shuffle random), tel que directement lisible dans le README.md du projet. Des améliorations possibles seraient d'utiliser des réseaux de neurones récurrent sur des features de mots plutôt que des features de documents.

Pour ce qui en est de l'identification de la langue, parmi 4 langues, l'erreur diminue de façon exponentiellement inverse en fonction du nombre de caractères lus, et atteint rapidement **100%**. La "demie-vie" (50% de bonnes prédictions) de l'exponentielle du meilleur classificateur est lorsqu'il aura vu ne serait-ce que 5 caractères, alors que la seconde demie-vie (75% de bonnes prédictions) arrive avec si peu que 10 caractères lus. Une amélioration possible serait un réseaux de neurones récurrent sur des 1-gram de caractères directement. Avec plus de données, la capitalisation des lettres serait probablement intéressante à conserver.

Finalement, les techniques de sélection des hyperparamètres pourraient être améliorées. Il aurait été bien d'effectuer une recherche random ou avec le Parzen Tree Estimator (TPE) algorithm plutôt que par grille. Une telle recherche pourrait par exemple être effectuée plus efficacement avec le RandomizedSearchCV de scikit-learn, ou avec la librairie Hyperopt pour ce qui concerne le TPE. D'autres librairies open-source sont intéressantes, telles que, par exemple, Spearmint et AutoML.

La méthodologie de cross-validation d'entraînement, et puis d'évaluation unique sur le test set suite à la cross validation pour évaluer le modèle choisi, fut respectée.

License

Le projet sera disponible publiquement sur GitHub sous cette license suivante une semaine suite à la remise, avec la permission du professeur.

Références

Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).
https://www.scipy.org/citing.html

Hunter, J. D., Matplotlib: A 2D graphics environment, (2007).
https://matplotlib.org/1.2.1/index.html

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12, 2825-2830 (2011).
http://jmlr.org/papers/v12/pedregosa11a.html

Rachel Tsz-Wai Lo, Ben He, Iadh Ounis, Automatically Building a Stopword List for an Information Retrieval System, JDIM, (2005).
http://terrierteam.dcs.gla.ac.uk/publications/rtlo_DIRpaper.pdf
https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature_extraction/stop_words.py

Bird, Steven, Edward Loper and Ewan Klein, Natural Language Processing with Python. O'Reilly Media Inc., (2009).
https://www.nltk.org/

Porter et al, NLTK: 2,400 stopwords for 11 languages, (2012)
http://nltk.org/book/ch02.html

Andrea Esuli, Fabrizio Sebastiani, SENTIWORDNET: A Publicly Available Lexical Resource for Opinion Mining In 5th Conference on Language Resources and Evaluation, pp. 417-422 (2006).
http://nmis.isti.cnr.it/sebastiani/Publications/LREC06.pdf

M. Marcus, B. Santorini and M.A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. In Computational Linguistics, volume 19, number 2, pp. 313–330, (1993).
https://dl.acm.org/citation.cfm?id=972470.972475

James Bergstra, Yoshua Bengio, Random Search for Hyper-Parameter Optimization, 13(Feb):281–305, (2012).
http://www.jmlr.org/papers/v13/bergstra12a.html

Luc Lamontagne et al., IFT-7022 Techniques et applications du traitement automatique de la langue (TALN): cours, énoncé de travail pratique 2, et données publiques (2018 ou avant).
http://www2.ift.ulaval.ca/~lamontagne/

Guillaume Chevalier, Python conv: a lightweight library to do for-loop-styled convolution passes on iterable objects, (2018).
https://pypi.org/project/conv/