Classification automatique: fake, true ou parodic ?

Problématique

Le but du projet est de catégoriser de manière automatique des articles de presse. Il s'agit de repérer si l'article en question est plutôt véridique, plutôt "fake" ou plutôt parodique.

Après avoir constitué le corpus, un algorithme permettant la classification automatique est lancé. Ici, nous utilisons l'algorithme d'apprentissage Keras, associé à la librairie numpy pour le traitement de données. Les données nécessitent un prétraitement, effectué en amont de l'utilisation de l'algorithme d'apprentissage.

La classification automatique est possible avec des données d'entraînement. Pour cela, un corpus d'entraînement a été constitué et annoté. Le corpus est constitué de plusieurs articles de presse provenant de sites web différents. Des balises ont été ajoutées autour du texte brut pour permettre de repérer le titre, l'url de l'article, le site de provenance, ainsi que sa catégorie: fake, trusted, parodic.

Notre objectif est d'observer si la variation de couche de neurones avec l'algorithme d'apprentissage de Keras influe sur la performance de la classification. De même, nous allons modifier certains paramètres pour constater des changements au niveau des résultats de précision.

Librairie et algorithmes

L'algorithme d'apprentissage utilisé est Keras. C'est une API de réseaux de neurones qui peut être associée aux librairies tensorFlow, ou CNTK. Ici, nous allons utiliser Numpy pour manipuler des matrices. Keras permet de créer des modèles d'apprentissage adapté à chaque situation. Il est conçu pour être simple d'utilisation, tout en gardant la possibilité de modifier les paramètres qui incombent à la tâche de classification automatique.

L'algorithme se découpe en deux parties: la partie d'entraînement train.py et la partie de prédiction predict.py. Le corpus d'entraînement est injecté à la partie d'entraînement qui va entraîner le modèle. Le corpus de test va ensuite être injecté à

la partie de prédiction qui se base sur le modèle obtenu précédemment pour prédire la classe de chaque document.

Nous utilisons également CNN (Réseau neuronal convolutif) comme le modele de réseau de neurons artificiels. Nous l'utilisons souvent pour traiter des images et des messages vocales, mais il peut aussi etre utiliser pour classer automatiquement des articles par catégories.

- Algorithme d'entraînement (train.py)

Tout d'abord, pour parcourir le corpus, nous avons dû utiliser la librairie ElementTree. Le corpus que nous analysons est au format XML, ElementTree nous permet de parser les éléments du corpus.

Ensuite, nous repérons les labels (classes) de chaque documents et nous les associons dans un fichier train_label.txt d'une part et test_label.txt d'autre part.
Une fois que les labels et les textes ont été identifiés, les paramètres sont définis. Notamment, la dimension des embeddings.

Les textes des documents sont ensuite tokenisés pour pouvoir repérer chaque mot.

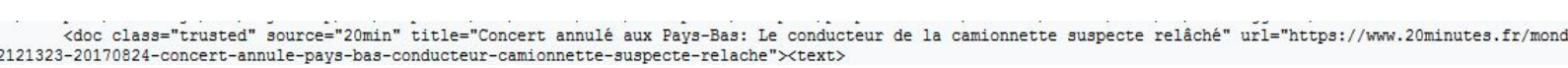
Les données sont divisées en training set, validation set et test set. Cela permet d'implémenter les abscisses et les ordonnées de chaque set. Le modèle est ensuite entraîné, puis sauvegarder.

- Algorithme de prédiction

L'algorithme de prédiction est assez simple. Il importe le modèle sauvegardé précédemment, et l'applique sur les données de test déjà traitées dans l'algorithme d'entraînement.

Features issues du prétraitement

Le corpus est d'abord prétraité. Comme expliqué dans la problématique, des balises ont été ajoutées autour du texte brut pour permettre de reconnaître les caractéristiques de l'article. La balise se présente ainsi:



Après cette balise <doc>, le texte brut est entouré de balises <text>.

Ensuite, TreeTagger est appliqué au texte brut de l'article. Le résultat obtenu est inséré entre les balises <treetagger>. Cela donne donc cette structure:

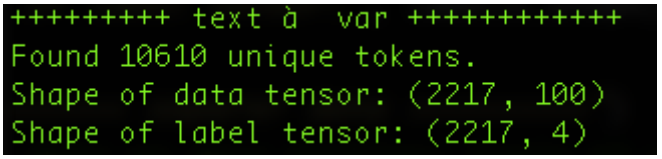
<doc class="" source="" title="" url="">
<text>
...
</text>
<treetagger>
…
</treetagger>
</doc>

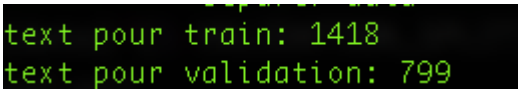
Après ces prétraitements, le corpus est prêt pour être analysé. Le Part Of Speech annoté à l'aide de treetagger va permettre à l'algorithme d'entraînement d'accroître la précision de la classification.

Statistiques

Avant prétraitement, le corpus est divisé en plusieurs documents, un document par étudiant. Au total, la taille des fichiers est de 703,09 Ko.

- Prétraitement 1 : nous prenons les contenus de chaque document et sa catégorie (fake, trusted ou parodic), puis nous coupons le contenu par phrase et ajoutons un label pour chaque phrase. Après prétraitement, le corpus a une taille de 2,57 Mo.
    - 155 articles au total, dont:
        - class="fake": 62 soit 40%
        - class="trusted": 55 soit 35,5 %
        - class="parodic": 38 soit 24,5 %
    - Au total:





- Prétraitement 2. Avec le corpus, nous nous concentrons sur les features, donc la taille de corpus est réduite de beaucoup.
    - Au total:

```
Found 14 unique tokens.
Shape of data tensor: (559, 100)
Shape of label tensor: (559, 4)
```

```
text pour train: 357
text pour validation: 202
```

Malgré la taille petite, le résultat avec que les features n'est pas mauvais par rapport au premier corpus que nous utilisions. Les résultats sont présentés dans la partie suivante.

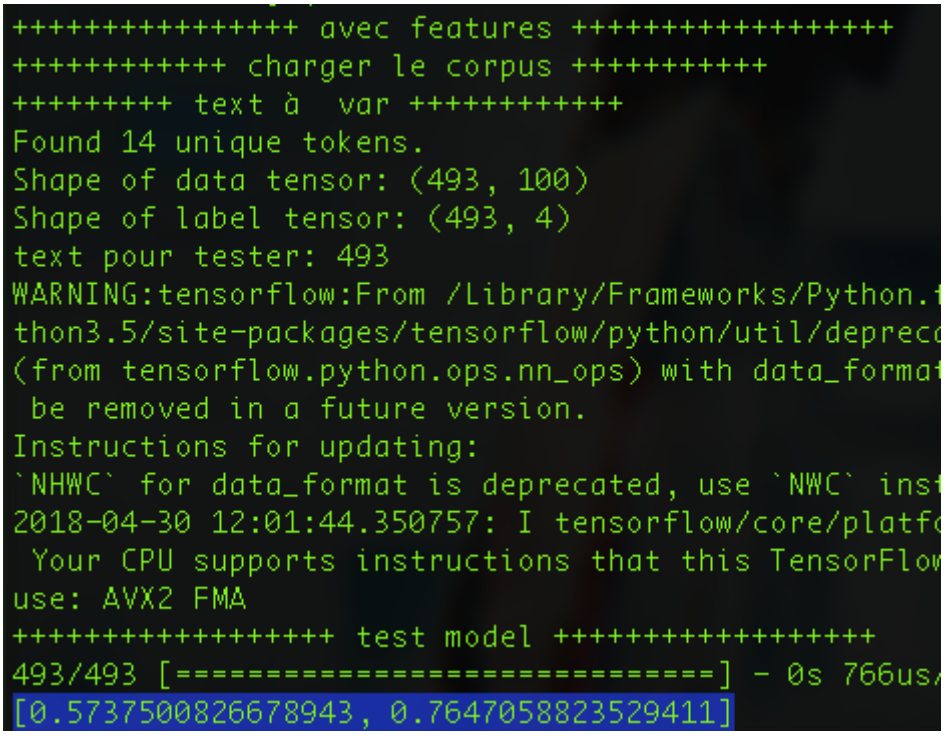
Résultats chiffrés

Sans features:

```
++++++++++++++++ sans features ++++++++++++++++++
++++++++++++ charger le corpus ++++++++++++
+++++++++ text à  var ++++++++++++
Found 10373 unique tokens.
Shape of data tensor: (2143, 100)
Shape of label tensor: (2143, 4)
text pour tester: 2143
++++++++++++++++++ test model ++++++++++++++++++
2143/2143 [==============================] - 2s 728us/step
[0.6982789917966777, 0.5669622021860683]
```

Présision: 0.5669622021860683

Avec features:



Précision: 0.7647058823529411

Discussion/amélioration

CNN a été très bon pour classer les articles, même si le corpus n'est pas très grand en terme de volume.

La limite de notre analyse est que nous n'avons pas utilisé le corpus étiqueté. Cela limite notre liste des features (nous avons ignoré les adverbes). Si nous construisions un corpus training avec les Part Of Speech, le résultat sera certainement plus performant.

Nous avons utilisé une seule couche de convolution car notre corpus est réduit. Avec un corpus de taille plus important, nous pouvons tester avec plusieurs couches de convolution.