Classifieur de titres utilisant les données du moteur de recherche ISIDORE et l'API Keras

Stéphane Pouyllau (0000-0002-9619-1002)[1]

[1]Ingénieur de recherche au CNRS, Huma-Num, Paris.

19 août 2020

Résumé

Ce document présente la création d'un classifieur de texte permettant de prédire l'appartenance disciplinaire de titres d'ouvrages ou d'articles en sciences humaines et sociales. Il met en œuvre ISIDORE et l'API Keras pour l'apprentissage profond (*Deep Learning*).

1 Introduction

Le but de ce document est de présenter la mise en œuvre d'un classifieur de titres utilisant les données d'ISIDORE [1] et l'API Keras [2] pour l'apprentissage profond (*Deep Learning*). Il décrit la construction du classifieur, son entraînement et son utilisation afin de prédire si des titres (d'ouvrages ou d'articles) peuvent être identifiés comme étant des documents d'histoire.

Le but n'est pas de faire ici un exemple exhaustif et détaillé utilisant toute la finesse de l'API Keras, mais de démontrer les potentialités et la relative facilité de l'apprentissage profond dans les SHS et de construire un exemple simple sous la forme d'une démonstration à destination des publics de sciences humaines et sociales [3]. Il s'appuie largement et met en œuvre sur le billet de blog *Practical Text Classification With Python and Keras*[1] et sur *How to Develop a Deep Convolutional Neural Network for Sentiment Analysis*[5]. Nous utilisons l'application Jupyter [4] pour inclure du code dans le document et cela nous permet également de mettre à disposition l'ensemble des données [5] et de les rendre exécutables directement avec Binder. Une bibliographie non exhaustive est disponible à la fin du document et guidera potentiellement la lectrice ou le lecteur.

2 Mise en œuvre : du choix des données d'entraînement à l'apprentissage

2.1 Préparation des données

Nous allons utiliser des titres de documents, issus ISIDORE lui-même, pour entraîner un classifieur à réseau de neurones utilisant les possibilités de l'API Keras. Cette API permet de créer, d'entraîner et d'utiliser des réseaux de neurones[2, p. 65]. La classification de texte, dans

1. Voir https://isidore.science
2. Voir https://keras.io
3. Je remercie ici Jean-Luc Minel, Professeur émérite à l'Université Paris Nanterre, de ses conseils lors de la création de ce classifieur.
4. Voir https://jupyter.org
5. Voir https://gitlab.huma-num.fr/spouyllau/isidore-jupyter

les réseaux de neurones, nécessite de vectoriser les titres pour pouvoir les traiter par la suite. Dans notre exemple nous utiliserons une représentation simple des données dit *Bag-Of-Words* (BOW) s'appuyant sur *CountVectorizer* de chez scikit-learn[3][1] [6].

Dans un premier temps, pour entraîner le classifieur à reconnaître de l'histoire, nous constituons un corpus d'entraînement à l'aide de la requête SPARQL ci-dessous [7]. ISIDORE catégorise plus de 7 millions de documents chaque mois venant de plus de 8000 bases de données au monde et ses données sont disponibles en RDF :

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
SELECT ?uri ?title where {
 ?uri sioc:topic <http://aurehal.archives-ouvertes.fr/subject/shs.XXXX>.
 ?uri dcterms:provenance <https://halshs.archives-ouvertes.fr>.
 ?uri dcterms:title ?title
 FILTER (lang(?title)='fr')
} LIMIT 2000
```

Ici, XXXX, représente les codes des disciplines utilisées dans ISIDORE pour catégoriser les documents. Il est possible de lister ces codes (qui sont des URI) avec la SPARQL suivante :

```
SELECT DISTINCT ?discipline WHERE {
 ?sujet <http://rdfs.org/sioc/ns#topic> ?discipline.
}
```

Ainsi, nous collectons 4000 titres de documents, répartis tels que :

— 2000 titres en histoire (code : shs.hist), étiquetés à 1

— 2000 titres qui ne sont pas de l'histoire (ici de l'infocom et de la sociologie), étiquetés à 0

La préparation du corpus d'entraînement est une phase essentielle du travail qui influence directement l'apprentissage du classifieur[4]. Nous ne gardons pas les mots grammaticaux (articles, prépositions) qui ne sont pas caractéristiques d'une discipline. Nous supprimons aussi les signes de ponctuation pour ne garder finalement que les mots des titres. Ainsi, notre corpus d'entraînement prend la forme suivante (extrait) :

```
Lutter ennemi interne  longue histoire  obsession  brésilienne;1
```

Nous chargeons le corpus d'entrainement dans un dictionnaire (qui est dans un fichier .txt) à l'aide de pandas :

[1]:
```
import pandas as pd

filepath_dict = {'histgeo':'MotsISIDORE-histoire-info-socio.txt'}

df_list = []
for source, filepath in filepath_dict.items():
    df = pd.read_csv(filepath, names=['words', 'label'], sep=';')
    df['source'] = source
    df_list.append(df)

df = pd.concat(df_list)
```

6. Voir https://scikit-learn.org/stable

7. ISIDORE propose l'ensemble de son corpus via une API et un SPARQL endpoint : https://isidore.science/sparql et https://isidore.science/sqe

```
print(df.iloc[0])
```

```
words     poèmes Lu You  végétarisme
label                             1
source                      histgeo
Name: 0, dtype: object
```

2.2 Préparation de l'entraînement et vectorisation

Il s'agit de définir le modèle et la vectorisation du corpus d'entraînement. Comme précisé ci-dessus, nous utilisons ici un BOW s'appuyant sur un encodage « 1 parmi n » (ou *One Hot encoding*) de chez sklearn dans ce document[5]. L'encodage *One hot* est une technique d'encodage binaire à un chiffre (0 ou 1) permettant d'encoder des données textuelles sous formes vectorielle[3, p. 165]. Nous présenterons par la suite, dans un second document, une approche utilisant un encodage en *Word Embedding*.

[2]:
```
from sklearn.model_selection import train_test_split

df_hist = df[df['source'] == 'histgeo']

sentencesA = df_hist['words'].values
y = df_hist['label'].values

sentences_train, sentences_test, y_train, y_test = train_test_split(
   sentencesA, y, test_size=0.25, random_state=100)

from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
vectorizer.fit(sentences_train)

X_train = vectorizer.transform(sentences_train)
X_test  = vectorizer.transform(sentences_test)
X_train
```

[2]:
```
<3000x7850 sparse matrix of type '<class 'numpy.int64'>'
        with 19264 stored elements in Compressed Sparse Row format>
```

2.3 Utilisation de l'API Keras pour l'entraînement

L'API Keras propose plusieurs modèles pour les réseaux de neurones. Nous allons utiliser le plus simple, le modèle *Sequential*[2, p. 68], en initialisant deux couches dont une couche cachée :

[3]:
```
from keras.models import Sequential
from keras import layers

input_dim = X_train.shape[1]

model = Sequential()
model.add(layers.Dense(50, input_dim=input_dim, activation='relu'))
model.add(layers.Dense(50, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
# 2 classes = binary_crossentropy
# + de 2 classes = categorical_crossentropy

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 50)                392550
_________________________________________________________________
dense_1 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 395,151
Trainable params: 395,151
Non-trainable params: 0
_________________________________________________________________
```

On lance l'apprentissage du classifieur, ici sur 10 *epochs* pour tester. Un *epoch* correspond à une itération où le modèle voit tout le corpus d'entraînement vectorisé pour apprendre.

[4]:
```
history = model.fit(X_train, y_train,
                    epochs=10,
                    verbose=1,
                    validation_data=(X_test, y_test),
                    batch_size=10)

loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Précision de l'entraînement: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Exactitude des tests: {:.4f}".format(accuracy))
```

```
Epoch 1/10
300/300 [==============================] - 1s 2ms/step - loss: 0.5301 -
accuracy: 0.7410 - val_loss: 0.4163 - val_accuracy: 0.7990
Epoch 2/10
300/300 [==============================] - 1s 2ms/step - loss: 0.1660 -
accuracy: 0.9413 - val_loss: 0.4506 - val_accuracy: 0.8190
Epoch 3/10
300/300 [==============================] - 1s 2ms/step - loss: 0.0505 -
accuracy: 0.9820 - val_loss: 0.5280 - val_accuracy: 0.8130
Epoch 4/10
300/300 [==============================] - 1s 2ms/step - loss: 0.0334 -
accuracy: 0.9880 - val_loss: 0.5785 - val_accuracy: 0.8140
Epoch 5/10
```

```
300/300 [==============================] - 1s 2ms/step - loss: 0.0261 -
accuracy: 0.9893 - val_loss: 0.6069 - val_accuracy: 0.8120
Epoch 6/10
300/300 [==============================] - 1s 2ms/step - loss: 0.0215 -
accuracy: 0.9910 - val_loss: 0.6437 - val_accuracy: 0.8080
Epoch 7/10
300/300 [==============================] - 0s 2ms/step - loss: 0.0214 -
accuracy: 0.9900 - val_loss: 0.6564 - val_accuracy: 0.8100
Epoch 8/10
300/300 [==============================] - 1s 2ms/step - loss: 0.0196 -
accuracy: 0.9907 - val_loss: 0.7136 - val_accuracy: 0.8040
Epoch 9/10
300/300 [==============================] - 1s 2ms/step - loss: 0.0195 -
accuracy: 0.9903 - val_loss: 0.6783 - val_accuracy: 0.8150
Epoch 10/10
300/300 [==============================] - 1s 2ms/step - loss: 0.0192 -
accuracy: 0.9897 - val_loss: 0.6966 - val_accuracy: 0.8060
Précision de l'entraînement: 0.9930
Exactitude des tests: 0.8060
```

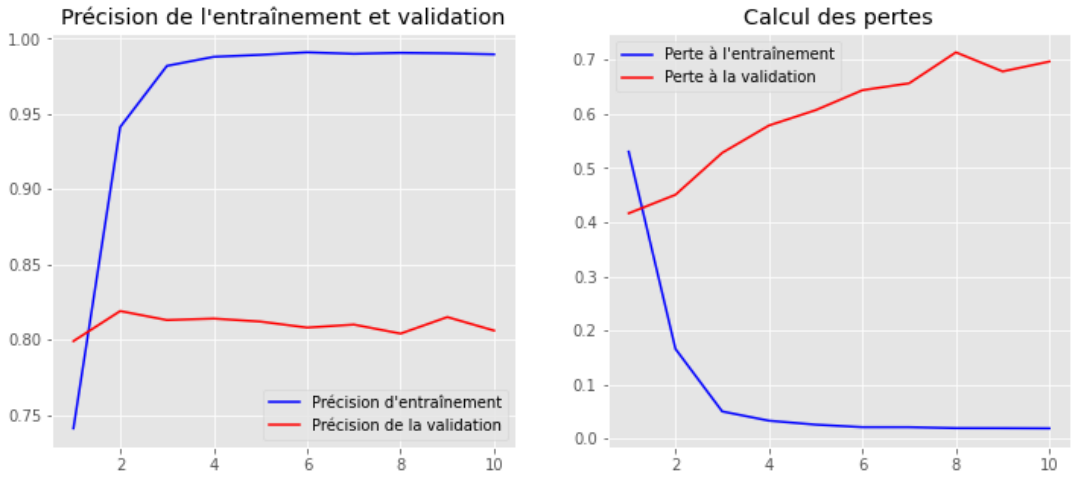
Nous affichons les statistiques courantes afin de visualiser les performances de l'entrainement :

[5]:
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
x = range(1, len(acc) + 1)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(x, acc, 'b', label='Précision d\'entraînement')
plt.plot(x, val_acc, 'r', label='Précision de la validation')
plt.title('Précision de l\'entraînement et validation')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(x, loss, 'b', label='Perte à l\'entraînement')
plt.plot(x, val_loss, 'r', label='Perte à la validation')
plt.title('Calcul des pertes')
plt.legend()
```

[5]:
```
<matplotlib.legend.Legend at 0x7fc6b8192e80>
```



L'apprentissage montre qu'il est probable que 3 *epochs* seraient suffisants pour entraîner le classifieur.

3 Utilisation du classifieur pour prédire l'appartenance à une discipline

On définit des titres d'articles à classifier et on les vectorise avant de les présenter au classifieur. Ces titres ne sont pas connus du classifieur, il ne sait pas s'il s'agit d'histoire ou pas :

[15]:
```
sentences_apredire = ['Stratégies éditoriales des musées. Une approche de la␣
↪médiation par l'accès ouvert aux données numérisées',
                      'Le monde karstique',
                      'Au plus près des âmes et des corps. Une histoire␣
↪intime des catholiques au xixe siècle',
                      'Cuba: pour une géographie du socialisme',
                      'Migrations en Turquie',
                      'Les autoroutes et informations',
                      'Les seigneuries et baronies au Moyen Âge',
                      'La guerre civile espagnole']
X_apredire = vectorizer.transform(sentences_apredire)
X_apredire = X_apredire.todense()
vectorizer.transform(sentences_apredire).toarray()
```

[15]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

Le tableau ci-dessus, réduit dans sa présentation, est le résultat de la vectorisation des titres qui seront présentés au classifieur (c'est-à-dire traité selon la représentation *One-hot* dans un BOW).

On utilise l'entraînement avec *model.predict_classes* de chez Keras pour prédire les articles qui sont des articles d'histoire. Les articles égaux à 1 sont sans doute des titres en histoire.

[16]:
```
predictions = model.predict_classes(X_apredire, verbose=1)
print()
print("Affichage des résultats :")
print()
for i in range(len(X_apredire)):
    if predictions [i] == 0:
        print("\"%s\" = %s ===> pas de l'histoire" % (sentences_apredire␣
 ↪[i], predictions [i]))
    else:
        print("\"%s\" = %s ===> sans doute de l'histoire" %␣
 ↪(sentences_apredire [i], predictions [i]))
```

```
1/1 [==============================] - 0s 2ms/step

Affichage des résultats :

"Stratégies éditoriales des musées. Une approche de la médiation par l'accès
ouvert aux données numérisées" = [0] ===> pas de l'histoire
"Le monde karstique" = [0] ===> pas de l'histoire
"Au plus près des âmes et des corps. Une histoire intime des catholiques au␣
 ↪xixe
siècle" = [1] ===> sans doute de l'histoire
"Cuba: pour une géographie du socialisme" = [1] ===> sans doute de l'histoire
"Migrations en Turquie" = [0] ===> pas de l'histoire
"Les autoroutes et informations" = [0] ===> pas de l'histoire
"Les seigneuries et baronies au Moyen Âge" = [1] ===> sans doute de l'histoire
"La guerre civile espagnole" = [1] ===> sans doute de l'histoire
```

Nous pouvons faire le constat que le classifieur fonctionne et qu'il a su reconnaître les titres qui relèvent de l'histoire.

4 Conclusion

Nous avons construit un classifieur « minimal » s'appuyant sur des données préparées et étiquetées afin de prédire la discipline de titres. Cette mise en œuvre a rencontré certaines limites et des choix ont été faits pour maintenir le but initial qui était de démontrer le potentiel de ces outils ainsi que la simplicité de la construction d'un tel classifieur.

Le choix de Keras comme API d'apprentissage profond pourrait être discuté. Plus largement, une analyse critique de ces instruments, selon les besoins des disciplines SHS, serait est très utile pour guider les développements des informaticiens et *data scientists*[6]. Une cartographie des différentes API d'apprentissage profond, régulièrement mise à jour et ciblant les besoins SHS, serait aussi d'une grande utilité. Elle permettrait de faire progresser l'utilisation raisonnée de ces techniques pour le traitement des corpus de données. Par ailleurs, les méthodes de recherche en SHS, qui s'appuient généralement sur le principe de l'accumulation des connaissances, sont un atout très important pour faire fonctionner des outils en apprentissage profond.

Comme nous l'avons vu, la souplesse de la mise en œuvre de Keras est aussi un atout très important pour créer rapidement des outils tout en se concentrant sur la préparation des données (entraînements, tests, etc.) qui doivent être au cœur de l'attention des chercheuses et chercheurs.

L'implémentation d'outils d'apprentissage profond, en complément de ceux déjà utilisés et relevant de l'apprentissage automatique (*Machine Learning*) ouvre de nouvelles perspectives

pour assurer la jouvence technologique et documentaire de dispositifs de recherche d'information tels qu'ISIDORE[7].

Références

[1] Janakiev, N., Practical Text Classification With Python and Keras, realpython.com (2020).

[2] Géron. A., Deep Learning avec Keras et TensorFlow, DUNOD, 2e ed. (2020).

[3] Ramsundar, B & al., TensorFlow pour le Deep Learning, O'Reilly (2018).

[4] Brownlee, J., How to Prepare Text Data for Deep Learning with Keras, machinelearning-mastery.com (2017).

[5] Brownlee, J., How to Develop a Deep Convolutional Neural Network for Sentiment Analysis (Text Classification), machinelearningmastery.com (2017).

[6] Boelaert, J & al., The Great Regression. Machine Learning, Econometrics, and the Future of Quantitative Social Sciences. Revue française de sociologie, Presse de Sciences Po / Centre National de la Recherche Scientifique (2018) 2018/3 (59), pp.475-506. hal-01841413

[7] Pouyllau, S., ISIDORE, un moteur et un assistant de recherche au service des enseignants chercheurs en lettres, sciences humaines et sociales, Octaviana (2020). 10670/1.vktg0u