A classifier using ISIDORE, the social and humanities search engine and Keras API for Deep Learning

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

[1]CNRS Research Engineer, Huma-Num, Paris.

August 21, 2020

Abstract

This document presents the creation of a text classifier to predict the disciplinary affiliation of titles (from books or articles) in the humanities and social sciences. It implements ISIDORE and Keras API for Deep Learning.

1 Introduction

The purpose of this document is to present the implementation of a title classifier using the data from ISIDORE[1] and Keras API[2] for Deep Learning. It describes the construction of the classifier, its training and use to predict whether titles (from books or articles) can be identified as history documents.

The aim here is not to make an exhaustive and detailed example using all the finesse of Keras, but to demonstrate the potential and relative ease of Deep Learning in the social sciences and humanities (SSH) and to construct a simple example in the form of a demonstration for SSH audiences[3]. It is largely based and implemented the blog post *Practical Text Classification With Python and Keras*[1] and *How to Develop a Deep Convolutional Neural Network for Sentiment Analysis*[5]. We use the Jupyter notebooks to include code in the document and all data sets are available and executable directly with Binder. A non-exhaustive bibliography is available at the end of the document and will potentially guide the reader.

2 Implementation: from choosing training set to learning

2.1 Prepare data

We will use document titles, from ISIDORE itself, to train a neural network classifier using the possibilities of Keras API. This API allows you to create, train and use neural networks[2, p. 65]. The classification of text in neural networks requires vectorizing the titles to be able to process them afterwards. In our example we will use a simple representation of the data called "Bag-Of-Words" (BOW) using *CountVectorizer* from scikit-learn[3][1][4].

First of all, to train the classifier to recognize "history" data, we constitute a training set using the SPARQL query below[5]. ISIDORE categorizes more than 7 million documents every

[1]See https://isidore.science

[2]See https://keras.io

[3]I would like to thank Jean-Luc Minel, Professor Emeritus at the University of Paris Nanterre, for his advice in creating this classifier.

[4]See https://scikit-learn.org/stable

[5]ISIDORE proposes an API and an endpoint SPARQL: https://isidore.science/sparql and https://isidore.science/sqe

month from more than 8000 databases worldwide and its data is available for the Linked Open Data in 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
```

Here, XXXX, represents the codes of SSH disciplines used in ISIDORE to categorize documents. It's possible to list these codes (which are URIs) with the following SPARQL:

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

We collect 4000 titles of documents, distributed as :

- 2000 titles in history (code: "shs.hist"), labeled at 1
- 2000 titles that are not history (here from "shs.info" and "shs.socio"), labeled at 0

The preparation of the training set is an essential phase of the work that directly influences the learning of the classifier[4]. We do not keep grammatical words (articles, prepositions) that are not characteristic of a discipline. We also remove punctuation marks to keep only the words of the titles. Thus, our training set takes the following form (excerpt) :

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

We load the training set (which is in a .txt file) using pandas to use it:

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

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

2.2 Prapare training and vectorization

This involves defining the model and vectorization of the training set. As specified above, we use a BOW based on a One Hot encoding from sklearn[5]. One hot encoding is a single-digit (0 or 1) binary encoding technique for encoding textual data in vector forms[3, p. 165]. We will then present, in a second paper, an approach using Word Embedding encoding.

[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 Use Keras for training data

The Keras API offers several models for neural networks. We will use the simplest one, the *Sequential* model"[2, p. 68], by initializing two layers including a hidden layer :

[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
_________________________________________________________________
```

We're initiating classifier training, here on 10 epochs to test. An epoch corresponds to an iteration where the model sees the entire training set for learning.

[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
Training accuracy: 0.9930
Test accuracy: 0.8060
```

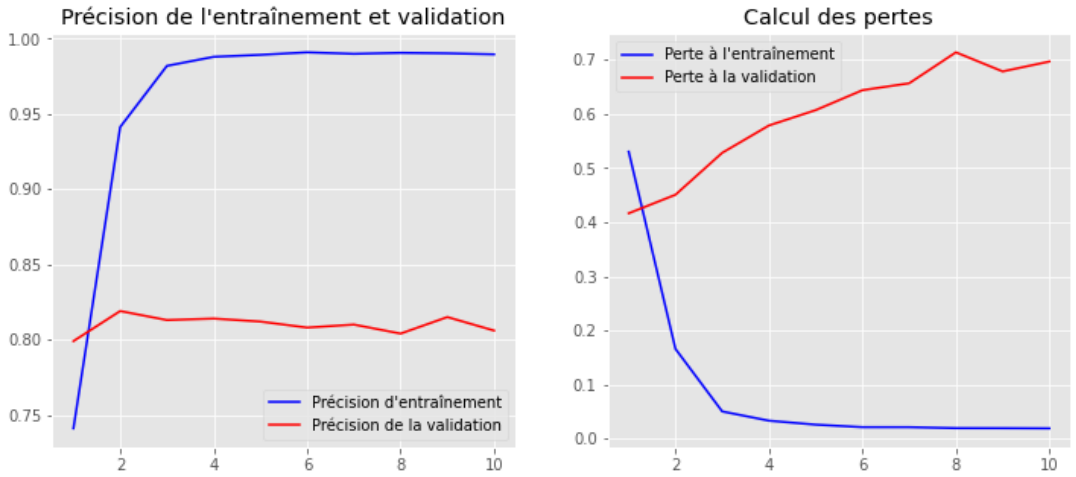
We display the current statistics in order to visualize the training performance:

[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>
```



Learning shows it's likely that three epochs would be enough to train the classifier.

3 Using the classifier to predict discipline

Titles to be classified are defined and vectorized before being presented to the classifier. These titles are not known to the classifier, he does not know if it is history or not:

[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]])
```

The above table, reduced in its presentation, is the result of the vectorization of the titles that will be presented to the classifier.

We use *model.predict_classes* from Keras to predict which articles are history. results equal to 1 are probably title from history.

[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

Results :

"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
```

We can see that the classifier works and that it has been able to recognize the titles that are part of history.

4 Conclusion

We constructed a "minimal" classifier based on prepared and labeled data to predict the discipline of titles. This implementation encountered limitations and choices were made to maintain the initial goal of demonstrating the potential of these tools and the simplicity of constructing such a classifier.

The choice of Keras as a Deep Learning API could be discussed. More broadly, a critical analysis of these instruments, according to the needs of the SSH disciplines, would be very useful to guide the developments of computer scientists and data scientists[6]. A mapping of the different Deep Learning APIs, regularly updated and targeted to SHS needs, would also be of great use. It would help to advance the reasoned use of these techniques for the processing of data corpuses. Furthermore, SSH research methods, which are generally based on the principle of knowledge accumulation, are a very important asset for making Deep Learning tools.

As we have seen, the flexibility of Keras' implementation is also a very important asset for quickly creating tools while focusing on the preparation of data (training, tests, etc.) that must be at the heart of the researchers' attention.

The implementation of Deep Learning tools, in addition to those already in use and related to machine learning opens new perspectives to ensure the technological and information rejuvenation of devices such as ISIDORE[7].

References

[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, machinelearningmastery.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