Using the Fasttext tool, for a better website search

Introduction:

In January 2024, during the second year of my studies in BTS SIO, I did a one month internship at the media group "Jeune Afrique Media Group". During my first weeks, I was given the task of continuing the exploratory work started by a developer, oriented in research with Fasttext using Python.

What is FastText ?

Fasttext is an open source library, developed by Facebook for the natural language processing(NLP), and automatic learning of this language. It is mainly used for text classification, and the creation of word representation models. Fasttext stands out by his capacity of training models quickly, even on large data sets

NLP and word representation models:

Natural Language Processing is a branch of artificial intelligence, focused on the interaction between computers and human language. The goal of this processing is to allow machines to learn, understand and be able to generate a language similar to the human's one, and understandable by humans.

These models capture ***semantic nuances***(subtle variations in meaning or significance that exist between terms: context, connotations), allowing for the utilization of the ***morphological subtleties in words*** (changes related to their grammatical construction), thereby enhancing the understanding of relationships between words. Here, we refer to "***embeddings***".

FastText deploys neural networks to generate word embeddings. These digital representations place words in a multidimensional space, capturing their **nuances** and **subtleties** (semantic and morphological). This approach allows the model to **improve** its understanding of language, revealing **subtle nuances** crucial in **understanding** difficult linguistic contexts.This is called:"***classification of words***"

How to use Fasttext ?: Installation

-It only need to install fasttext, by following the official documentation : https://fasttext.cc/docs/en/support.html

-Or simply, create a local copy of the fasttext GitHub repository:

```
stephane@OptiPlex-7070:~/Téléchargements/fasttextInstall$  git clone https://github.com/facebookresearch/fastText.git
Clonage dans 'fastText'...
remote: Enumerating objects: 3986, done.
remote: Counting objects: 100% (1045/1045), done.
remote: Compressing objects: 100% (182/182), done.
remote: Total 3986 (delta 920), reused 882 (delta 859), pack-reused 2941
Réception d'objets: 100% (3986/3986), 8.29 Mio | 13.05 Mio/s, fait.
Résolution des deltas: 100% (2527/2527), fait.
```

Just enter into the new folder, than use the command : make
(this will automatize the compilation process, with a folder containing a "*Makerfile*" file, this command execute the instructions inside this file):

```
stephane@OptiPlex-7070:~/Téléchargements/fasttextInstall$ cd fastText/
stephane@OptiPlex-7070:~/Téléchargements/fasttextInstall/fastText$ make
c++ -pthread -std=c++17 -march=native -O3 -funroll-loops -DNDEBUG -c src/args.cc
c++ -pthread -std=c++17 -march=native -O3 -funroll-loops -DNDEBUG -c src/autotune.cc
c++ -pthread -std=c++17 -march=native -O3 -funroll-loops -DNDEBUG -c src/matrix.cc
c++ -pthread -std=c++17 -march=native -O3 -funroll-loops -DNDEBUG -c src/dictionary.cc
```

Then, you just have to install the python module for fasttext:

```
stephane@OptiPlex-7070:~/Téléchargements/fasttextInstall/fastText$ sudo python3 setup.py install
/usr/lib/python3/dist-packages/setuptools/dist.py:723: UserWarning: Usage of dash-separated 'description-fi
nstead
  warnings.warn(
running install
/usr/lib/python3/dist-packages/setuptools/command/install.py:34: SetuptoolsDeprecationWarning: setup.py ins
  warnings.warn(
/usr/lib/python3/dist-packages/setuptools/command/easy_install.py:158: EasyInstallDeprecationWarning: easy_
  warnings.warn(
```

You also need to have Python3 installed, use the command:“ ***sudo apt install python3*** “
The installation is now complete !
.

Creation and use of a modele:

The goal of this word classification is to associate a document (newspaper articles, their summaries and titles) to one or many categories, and thereby create ‘classifiers' gathering all the datas(categories + contents). Fasttext learning allows the creation of those classifiers(using : context, connotation,...), but their creation ***requires labeled data(classifiers)***, here a category is denoted by the designation ***label***.

Thereby, before create a model, label data is essential, and for that we need to write a line, starting with the labels, followed by the content that will be labeled:

-We label the categories by adding the prefix : “***__label__***”, followed by the category, and we repeat this format for every label of a document.

“” __label__Number1 __label__Number2 Content_of_the_Document “”

Now that our data are in the right format, we can create a model under Python, by using the file containing our data:

```
1   import fasttext
2   model = fasttext.train_supervised(input="JA_labels.txt")
```

We import the fasttext library, and here the "***input***" is indicating the file containing our labeled data
By executing the code, we get:

```
stephane@OptiPlex-7070:~/Bureau/elasticsearch_nlp$ /bin/python3 /home/stephane/Bureau/elasticsearch_nlp/src/pythonDocumentationFAstText.py
Read 0M words
Number of words:  32516
Number of labels: 1425
Progress: 100.0% words/sec/thread:   91290 lr:  0.000000 avg.loss: 11.705429 ETA:   0h 0m 0s
```

Informations about the new model are sent to the user: number of words and the number of labels in the file.
We can than ask a question to our model, using the method : "***model.predict***", followed by our question:

```
print(model.predict("Qui est le président de la Côte D'Ivoire?"))
```

The question is :" ***Who is the president of Côte d'Ivoire ?***"

```
Progress: 100.0% words/sec/thread:   88298 lr:  0.000000 avg.loss: 11.957835 ETA:   0h 0m 0s
(('__label__18',), array([0.35248792]))
```

This method predicts a label, here the label is the 18, equivalent to the category: "West Africa".

```
"18": "Afrique de l'Ouest",
```

Côte d'Ivoire is indeed a country of West Africa.

Parameters:

With fasttext we can add parameters at our model, like:

- ***k***: Allows to define the number of labels we want in return,
- ***epoch*** : The number of times each examples is seen, can be increase with this parameter, and allow a better learning of the model
- ***lr***: The learning rate can be modified with this parameter, between 0 and 1.0, where 0 means that the model isn't learning.
- ***wordNgrams***: In fasttext, initially a token is a word, and the number of words constituting the token can be changed. This parameter allows the model to **learn a token by group of words**. In our case, we want to put this parameter at '**2**'(initially 1), which means that our model will learn by group of two words.

This parameter notion is useful, because it allows us to **get better results** than the initial model(without parameters).

Autotune(Automatic Hyperparameter optimization):

By using FastText's autotune feature, we can automatically find the optimal hyperparameters for our dataset. But, this feature requires a labeled data file, which we assure the content, and will play the role of : "***autotuneValidationFile***".

```
model = fasttext.train_supervised(input="JA_labels.txt", autotuneValidationFile="JA_labels_autotune.txt")
```

We obtain this result:

```
stephane@OptiPlex-7070:~/Bureau/elasticsearch_nlp$ /bin/python3 /home/stephane/Bureau/elasticsearch_nlp/src/pythonDocumentationFAstText.py
Progress: 100.0% Trials:   17 Best score:  0.112360 ETA:   0h 0m 0s
Training again with best arguments
Read 0M words
Number of words:  32516
Number of labels: 1425
Progress: 100.0% words/sec/thread: 18[illegible] lr: 0.0[illegible] avg.loss: 17.0[illegible] ETA:   0h 0m 0s
```

Here, the model identified [illegible] optimal parameters, then [illegible] with these parameters.

A method 'test' also exist, it allows us to test the validity of the labeled data, and shows us the model precision and the number of labels correctly predicted (recall) by the model:

```
Progress: 100.0% words/sec/thread:  165690 lr:  0.000000 avg.loss: 17.034775 ETA:   0h 0m 0s
(15, 0.2, 0.04054054054054054)
```

For this test:

- ➔ précision: 0.2 (not really convincing, but this means that we can improve our model, here we only have 15 labels)
- ➔ recall:0.04

All the essential elements to the understanding of this library have been mentioned.

Goals and Realisations

Goals:

- Continuing the exploratory work about the vectorization part of fasttext (***embeddings***) started by a developer, and using this code basis to deepen the label and autotune part.
- Make the fasttext tool viable for a future implementation on the website.

Realisations:

- I created a file to store my functions that will shape the data as we want with fasttext (from a JSON format to a Python List), ainsi que de les exploiter
- Labeled file with all Jeune Afrique articles from 2010 to 2024
- Use the autotune feature + create a file by hand, where we guarantee the content (labels and contents linked).
- Tests of many models with different parameters, and I got a potentially usable model for the newspaper, but unoptimised, nor further trained due to a lack of resources.

Format the data:

I started my part on this project with a JSON file regrouping 5000 articles of Jeune Afrique, which I had to exploit. My first obstacle was to exploit this data, so I used functions from the "***json"*** library to transform my JSON file into a Python list and string. :

- "***json.load***" : allows you to take a JSON file, and return it as a Python List.

```python
def fichierJSON_vers_ListePython(cheminFichierJSON):
    with open(cheminFichierJSON, 'r', encoding='utf-8') as fichier_json:
        listePython = json.load(fichier_json)
    return listePython
```

- “***json.loads***” : take a JSON string, and return a Python Dictionary.

```
liste_python[indice]['tags'] = json.loads(liste_python[indice]['tags'])
```

In our case, she turns strings where our **labels** are, into a **dictionary** of the shape ***“key : value”***, where our **value** is the label title..

Now that we can extract our data from a Python List, we can focus on the automatic add of the prefix "***__label__***" for our labels, followed by the concerned content.

⚠ Warning ! ⚠: You must first clean your data, in my case, the articles are taken from the website Jeune Afrique and have many HTML tags, so we must eliminate these “**parasitic**” elements.

```
def ecriture_label(ID_Tag, fichier):
    print(f"__label__{ID_Tag} ", end="")
    fichier.write("__label__" + str(ID_Tag) + " ")
```

This function allows us to write in a file our prefix, followed by our labels. She write in "***fichier***", with the method "***write***", and this method is only usable if we open "fichier" in **writing** mode, with : " ***with open(fichier, 'w')*** "

The function“***fasttestpython***” have in parameters our Python List containing our data and the file where our labeled dataset will be written. This function returns nothing, but changes the file entered as a parameter.

```
def fasttestpython(liste_python, fich
    transforme_str_en_liste(liste_pyt
    with open(fichiertxt, 'w') as fic
        for indice in range(len(liste
            lister_elements_fichier(l
            lister_elements_fichier(l
            lister_elements_fichier(l

            ecriture_titre(liste_pyth

            ecriture_resume(liste_pyt
            print("")
            fichier_ja.write("\n")
```

The functions “***ecriture_titre***” and “***ecriture_resume***” allow writing into the file “fichier”, the same way as “***ecriture_label***”, but for the summaries and the titles.

The function “***lister_elements_fichier***” will call the function “***ecriture_label***” for every label linked to a content.

We notice that we open our file in "writing mode", thereby all the calls to the "write" method will be executed
.
After the call at the function "***fasttestpython***", we obtain a usable file text by fasttext, which is named : "***JA_labels_2010_to_2024***.txt". This is the file entered as a parameter, which has been modified by this function.

I have also created a file text named : ***"JA_labels_autotune.txt"***, written by hand, it will play the role of the ***"autotuneValidationFile"***.

Tests performed:

- Supervised model with manually entered parameters
    - Unclear answer, with 2 out of 5 labels matching our search
- Model using the autotune feature(automatisation of our parameters)
    - More accurate answer, 3 labels out of 5 match our search, with a learning time of one hour
    - Learning time cannot be increased, due to lack of resources (insufficient RAM)

Thought on the project:

This project was specifically designed for Jeune Afrique, which means that our functions, presented above, are only compatible with the files extracted from the Jeune Afrique database.

Regarding our articles and the veracity of their tags, we assume that our articles are well tagged, they will serve as a reference for our model. Finding a problem in our data is possible, when I wrote the file JA_labels_autotune.txt, I noticed an article mislabelled (the label "Coup d'État au Gabon", was linked to an article on the architecture of the store "Tati" in Barbes, Paris).

Continuity of the project:

The results obtained are encouraging, and seems to show that this project will be deepened on the learning part, with a computer having more ressources (**more than 8GB of RAM is essential**).

It will also be possible to integrate this tool in the search of the Jeune Afrique site, which will allow users to search for an article in the database using the tags returned by fasttext.

This project allowed me to discover the notions of ***Machine Learning*** (***NLP***, ***embeddings***), to ***extract data*** and to ***exploit*** these data for my mission of deepening in the autotune functionality and the labels.

Currently in BTS SIO, I want to guide the rest of my studies in the fields of AI, machine learning and data processing. Working on this project was an opportunity to open myself to these notions.