How I learned to stop worry and trust the script
Or how to automatically create ad hoc text classification model in python

The internet is great, we can actually find information about any subject. The difficult part is then to organise them and classify those data in the right bucket given the topic you are interested in. A classical approach would be to go through everything, read it, store the topic you extracted somewhere somehow (in your head, post-it, csv file…), and then classify manually all those documents in the right bucket. It can be somehow ok if your corpus is not too big, but once you reach a dozen documents whose length may vary a lot, it becomes quite a burden. Moreover, you may be interested in only one of the topic covered in the document and would still have to read the whole bunch and your classification may not be consistent. Surelly there is a better way to use our time…
Keeping those points in mind, I programmed a script for automatic extraction and classification of corpus of text data. Apart from creating this solution, the others objectives were to get some hands on experience with data scrapping, APIs and spacy.

1) Finding the data :

For any data science project, the most difficult part is often to find a meaningfull dataset with enough content and representative data to actually perform well. Lucky for us, at least for NLP projects, a lot of people have willingly published around 30 millions of articles in close to 300 langages. For the last 18 years people freely published and corrected each others on an open online platform. I am of course refering to the online encyclopedia wikipedia.
The platform offers a well documented, quite reliable and well organised encyclopedia on many subjects and could therefore be used as a reference for data extraction for NLP projects.

First, the data have to be scrapped from wikipedia. Asthe wikipedia pages have this format https://en.wikipedia.org/wiki/Turing_machine, where ‘en’ stands for english and ‘Turing_machine’ is the subject you want to explore, you can dynamically create some urls that you will use to get the data (at first I wanted to use the wikipedia API but when creating this script none was supported). Doing so, the script will go through the html page and exctract each substitle in a span section and use it later as labels for the dataset.

This is done using beautifulsoup via this function :

```
####################
## this function takes as input the language and the subject you want to create a classifier for
## it will then get the ad hoc url in wkipedia
## then it will scrape each span to get its title as a category
####################
def get_category(language, subject):
 wikipedia.set_lang(language)
source = urllib.request.urlopen(wikipedia.page(subject).url).read()
soup = bs.BeautifulSoup(source,'lxml')
soup_txt = str(soup.body)
category = []
for each_span in soup.find_all('span', {'class':'mw-headline'}):
soup = BeautifulSoup(str(each_span).replace(' ','_'), "html.parser").getText()
category.append(soup)
```

return category

Then, once you have the labels, you need the data that go with. This is done via

```
####################
## this function takes as input the language and the subject
## it gets the text for each span and cleans it
## then it will output a list containing each clean sentence
####################
def get_data(language, subject):
wikipedia.set_lang(language)
source = urllib.request.urlopen(wikipedia.page(subject).url).read()
soup = bs.BeautifulSoup(source,'lxml')
soup_txt = str(soup.body)
div = []
for each_span in soup.find_all('span', {'class':'mw-headline'}):
str(each_span).replace(' ','_')
div.append(str(each_span))
filter_tag = []
i = 0
while i < len(div)-1:
start = div[i]
end = div[i+1]
text = soup_txt[soup_txt.find(start)+len(start):soup_txt.rfind(end)]
soup = str(BeautifulSoup(text, "html.parser"))
soup = re.sub("([\(\[]).*?([\)\]])", "\g<1>\g<2>", soup)
soup = BeautifulSoup(soup, "html.parser")
soup = re.compile(r'<img.*?/>').sub('', str(soup.find_all('p')))
soup = BeautifulSoup(soup, "html.parser")
soup = (re.sub("[^a-zA-Z,.;:!0-9]","
",soup.getText()).replace('[',").replace(']',").lstrip().rstrip().lower())
clean_text = re.sub(' +', ' ',soup).replace(',',' ')
filter_tag.append(clean_text)
i += 1
return filter_tag
```

Finally, everything is then put together via the functions :

```
####################
## this function takes as input the list of filter tags
## it will output a list containing the number of sentences in each category
####################
def get_len_list(filter_tag):
```

```
filtered_text = []
len_list = []
i = 0
while i < len(filter_tag):
doc = nlp(filter_tag[i])
text = [sent.string.strip() for sent in doc.sents]
filtered_text.append(text)
len_list.append(len(filtered_text[i]))
i += 1
return filtered_text,len_list
```

```
####################
##tegory this function takes as input the len_list and the list of category
## it will output a dataframe containing the label (category) for each sentence
####################
def generate_dataset(len_list, category):
i = 0
label_list = []
while i < len(len_list):
j = 0
if len_list[i] != 0:
while j != len_list[i]:
label_list.append(category[i].lower())
j += 1
i += 1
flat_list = [item for sublist in get_len_list(get_data(language, subject))[0] for item in sublist]
data = {'text': flat_list,'label': label_list}
df = pd.DataFrame.from_dict(data)
print(df.head())
print('Repartition of labels:', df['label'].iloc[0])
print('Data Shape:', df.shape)
return df
```

that will create an had hoc pandas dataframe based both on the subject and the langage you want your classifier to be created for. Each sentence extracted in each span is extracted and the subtitle it was linked to is used as a label.

2) Training a model :

Once you have your ad hoc dataset with its label, it is now time for picking a model and train it. At first, I wanted to experiment with LSTM for classification, but the results were not very good, I suppose due to the fact that I did not have a lot of data. So I chose to go for a more classical approach using some well known NLP manipulations.

First, I got rid of the stop words for the given langage (spacy proposes built-in lists for english, french, german, spanish, portuguese, italian and dutch). Then, I applied lemmatizatio to keep only the root of the word. I could have used stemming, but it does not fit well for latin langages. I applied a bit more manipulations to get a cleaner dataset, fit for NLP manipulations. The issue is that models understand numbers, not words, so I decided to use a bag of words approach to transform those words to numbers and later applied a grid search on a random forest to get the best model possible for this given subject in this given langage.

The model is then saved in pickle format and can be called for classification.

3) Conclusion :

This little project gave me the opportunity to try a lot of different new concepts and technologies from classic NLP manipulations using the latest libraries to web scrapping and public APIs manipulations and other libraries like argparse or pickle. I would like to keep improving on this by adding virtual environment and docker container on this to make them easier to reproduce and share as well as experimenting for good with LSTM on this subject.
But at least with this project I have a consistent and quite powerful way to create automatic models for text corpus classification on a given subject in 7 langages. And this is how I learned to stop worrying and trust the script.

the repository can be found here : https://github.com/elBichon/blue_orchid