# Wikipedia classifier

Louis Foucard

---

# Description

This python classifier can take in any number of Wikipedia category, scrap through the articles and subcategory to collect text in each category, construct the term document matrix and fit a stochastic gradient descent classifier. The machine learning python library sklearn is used to preprocess the mined text and to generate/fit the model.

There are 3 files:

- **wikipedia_scrapper.py**: python module that uses BeautifulSoup and urllib2 to recursively scrap through the specified categories. Takes as input a list of Wikipedia categories, and the maximum level of subcategories to search.

- **build_wiki_classifier.py**: main python script that collects the text using the scrapper, builds the model and creates a classifier report that describes the model's performance. It takes as input a text file with any number of Wikipedia categories followed by the maximum level of subcategory, and the names under which to save the model.

- **category_predicter.py**: python scripts that loads the serialized model and predicts the category (and probability for each category) for a Wikipedia url. It takes as input the names of the model and the url to classify.

The text mined from the Wikipedia categories is first preprocessed by removing stop words, punctuation and by generating the term frequency–inverse document frequency matrix. Once the term frequency matrix is built, a classifier model is fitted.

## Usage

To build the classifier (15-30mn depending on internet connection and subcategory number):

```
python build_wiki_classifier.py <categories_file_name> <model_save_name>
<vectorizer_save_name>
Example: python build_wiki_classifier.py  categories.txt model vectorize
```

To run the classifier:

```
python category_predicter.py  <categories_file_name> <model_name>  <vectorizer_name>
<url>
```

```
Example: python category_predicter.py categories.txt model vectorizer
https://en.wikipedia.org/wiki/Mycotic_aneurysm
```

# Dependencies

BeautifulSoup, urllib2, re, sklearn, numpy, nltk

# Analysis

### Text scrapper

One of the parameters of the Wikipedia scrapper is how deep into subcategories should the scrapper be allowed to go. If that number is too high, the subject of the resulting articles will start to diverge very fast, but if it is zero, there is not enough data to analyze. Here I allowed the scrapper to go into one level of subcategory, but no deeper. That parameter can be specified for each category in the input.

### Preprocessing

Different preprocessing techniques were tried to reduce dimensions of the problem: stemming can be applied optionally, but it did seem to improve the result, and is rather slow. Singular value decomposition to reduce the dimension of the term document matrix was also tried but did not improve the final result.

### Classifiers

Several classifiers were tried such as Random Forest or Naïve Bayes. The best scores were obtained using a stochastic gradient descent with a logistic regression loss or a linear support vector classifier with hinge loss. The logistic regression classifier was chosen to compute the probabilities of belonging to each category. The rest of the parameters were optimized using gridsearch.

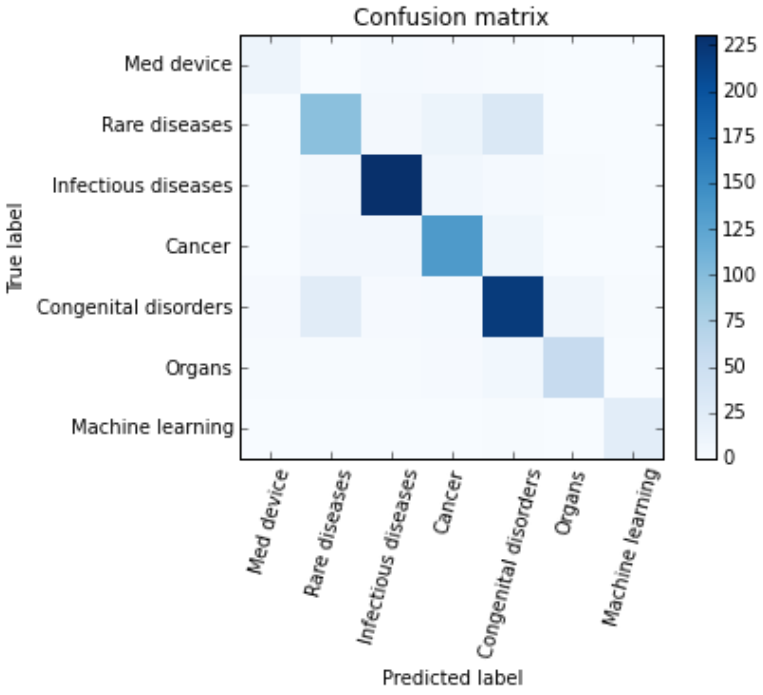
### Performance

To estimate how well the classifier will generalize to new articles in each category, the text scrapped from the Wikipedia categories was split into training data (80%) and testing data (20%), not used to fit the model. Below is a summary of the performance of the model on the testing data:

| | *precision* | *recall* | *f1-score* | *support* |
|---|---|---|---|---|
| *Medical devices* | 0.95 | 0.75 | 0.84 | 28 |
| *Rare diseases* | 0.75 | 0.68 | 0.71 | 213 |
| *Infectious diseases* | 0.90 | 0.94 | 0.92 | 333 |
| *Cancer* | 0.87 | 0.92 | 0.89 | 177 |
| *Congenital disorders* | 0.80 | 0.82 | 0.81 | 346 |
| *Organs* | 0.89 | 0.86 | 0.88 | 86 |
| *Machine learning* | 1.00 | 0.96 | 0.98 | 47 |
| avg / total | 0.85 | 0.85 | 0.85 | 1230 |

We obtain an average of 0.85 for the f1 score (combination of precision and recall) across all categories. We can see that the machine learning category has the highest f1 score: this is to be expected, since it is a category very different from all 6 others, with a very different vocabulary.

The category with the lowest f1 score is the rare disease categories. This might be due to the fact that it is a rather broad category: rare diseases may take many forms, such as infections, cancers, congenital disorders, and affect many organs. To check whether this category is often confused for another one specifically, let's look at the confusion matrix below:




Here we can see that the rare disease and the congenital disorder categories are the closest ones, and are often confused for one another. This might be due to the fact that congenital diseases are also quite rare (<5% of the population), and rare diseases might include congenital diseases. On the other hand, topics dealing with cancer or organs in general are much broader and might for example use the word "rare" a lot less often, which results in more clearly separated categories.

In fact, the confusion between the rare disease and congenital disorders is what brings the score down the most. Removing either category before training leads to f1 scores of 0.92+ across all remaining categories.

The most informative features for each category can be found by looking at the highest coefficients in the classifier. Here are the 5 most informative feature for each category, in increasing order:

```
Medical_devices: stent delivery company devices device

Rare_diseases: autosomal patients syndrome disease rare
```

```
Infectious_diseases: medicine infectious veterinary virus infection

Cancer: anticancer research tumor oncology cancer

Congenital_disorders: disabilities gene genes genetic congenital

Organs_(anatomy): kidney gland glands brain organ

Machine_learning_algorithms: decision data learning mining algorithm
```

A sanitary check is to verify that the most informative feature for each category appears in the name of the category, which is the case here.

## Possible improvements

The level of subcategory allowed for each category could be found by using a previously trained classifier: if the subject of the subcategory seems close enough to the original category, the scrapper should be allowed to mine it. Another possible improvement is to use deep learning: an LSTM could be trained on sequences of words to get more context from the data instead of relying on single words.