ZIPF CLASSIFICATION OF TEXTS

Prof. Alberto Borghese
6 CFU

Luca Cappelletti

Course Project
Year 2017/2018

IT Master Degree
Universiy of Milan
Italy
30 giugno 2018

Zipf Classifier

Using Zipfs to classify books and articles

Introduction

Hello, I'm Luca Cappelletti and here I will show you a complete explanation and example of usage of ZipfClassifier, a classifier that leverages the assumption that some kind of datasets (texts, some images, even sounds in spoken languages) follows the Zipf law.

In the following examples, we will be trying to classify books to their authors and style periods and articles to their type of content.

How to use this notebook

This is a Jupyter Notebook. You can either read it here on github or, **preferably** to enjoy all its features, run it on your own computer. Jupyter comes installed with Anaconda, to execute it you just need to run the following in your terminal:

```
jupyter-notebook
```

What we will use

The packages

We will use obviously the ZipfClassifier and other two packages of mine: Zipf (comes installed as a dependecy in the zipf_classifier) to create the distributions from the texts and Dictances for the classifications metrics, even though any custom metric is usable. If you need to install them just run the following command in your terminal:

```
pip install dictances zipf_classifier
```

In [1]:

```
from dictances import (hellinger, intersection_squared_hellinger,
                       intersection_total_variation, jensen_shannon,
                       kullback_leibler, normal_total_variation,
                       squared_hellinger)
from zipf.factories import ZipfFromDir

from zipf_classifier import ZipfClassifier
```

Additional packages

We will also be using some utilities, such as the loading bar `tqdm`. If you don't have them already you can install them by running:

```
pip install tqdm tabulate
```

The others packages should be already installed with python by default.

In [2]:

```
# To import source code from external packages
import inspect
# To use mathematical functions and defines
import math
# To access os functions on files and directories
import os
# To randomize (predictably) datasets
import random
# To access high level os functions on files and directories
import shutil

# To output html tables
import tabulate
# To display jupyter notebook html content (such as tables)
from IPython.display import HTML, display
# To highlight code
from pygments import highlight
from pygments.formatters import HtmlFormatter
from pygments.lexers import PythonLexer
# A nice loading bar for showing progresses
from tqdm import tqdm_notebook as tqdm
```

Some small helpers

Let's make ome small functions to help out loading folders:

In [3]:

```
def get_dirs(root):
    """Return subdirectories under a directory."""
    return [
        root + "/" + d for d in os.listdir(root)
        if os.path.isdir(root + "/" + d)
    ]
```

and the book folders:

In [4]:

```
def get_books(root):
    """Return all books found under a given root."""
    return [
        book[0] for book in os.walk(root) for chapter in book[2][:1]
        if chapter.endswith('.txt')
    ]
```

and the saved zipfs:

In [5]:

```
def get_zipfs(root):
```

```
    """Return all zipfs found under a given root."""
    return [
        zipfs[0] + "/" + zipf for zipfs in os.walk(root) for zipf in zipfs[2]
        if zipf.endswith('.json')
    ]
```

Some stylers

In [6]:

```
frame_number = 30
```

In [7]:

```
def b(string):
    """Return a boldified string."""
    return "\033[1m%s\033[0;0m" % string
```

In [8]:

```
def red(string):
    """Return a red string."""
    return "\033[0;31m%s\033[0;0m" % string
```

In [9]:

```
def yellow(string):
    """Return a yellow string."""
    return "\033[0;33m%s\033[0;0m" % string
```

In [10]:

```
def green(string):
    """Return a green string."""
    return "\033[0;32m%s\033[0;0m" % string
```

In [11]:

```
def gray(string):
    """Return a gray string."""
    return "\033[0;37m%s\033[0;0m" % string
```

In [12]:

```
def print_function(function):
    """Print the source of a given function."""
    code = inspect.getsource(function)
    formatter = HtmlFormatter()
    display(
        HTML('<style type="text/css">{}</style>{}'.format(
            formatter.get_style_defs('.highlight'),
            highlight(code, PythonLexer(), formatter))))
```

In [13]:

```
def success(results, metric):
    """Show the result of a given test."""
    successes = results["success"]
```

```
    total = successes + results["failures"] + results["unclassified"]
    percentage = round(successes / total * 100, 2)
    if percentage > 85:
        metric_name = green(metric.__name__)
    elif percentage > 70:
        metric_name = yellow(metric.__name__)
    else:
        metric_name = red(metric.__name__)
    print(
        "Success with metric %s: %s" % (metric_name, b(str(percentage) + "%")))
    display(
        HTML(
            tabulate.tabulate(
                list(results.items()), ["Info", "Values"], tablefmt='html')))
```

The datasets

I've prepared three **datasets**:

Authors dataset

Dataset of english books from three **famous authors**: **D. H. Lawrence**, **Oscar Wilde** and **Mark Twain**.

This dataset will be used to build a classifier able to classify the books to the respective author.

Periods dataset

Dataset of english books from four **style periods**: **Modernism**, **Naturalism**, **Realism** and **Romanticism**. This dataset will be used to build a classifier able to classify the books to the respective style period.

Recipes dataset

Dataset of italian articles, some containing recipes and some containing food reviews, food descriptions (eg wikipedia) or other articles.

We will use this to classify articles to **recipes** and **non recipes**.

Retrieving the datasets

We download and extract the datasets:

- Link to authors dataset
- Link to periods dataset
- Link to recipes dataset

Put them in the same folder of this notebook to use the datasets.

In [14]:

```
datasets = ["authors", "periods", "recipes"]
```

Before going any further, let's check if the dataset are now present:

In [15]:

```
for dataset in datasets:
    if not os.path.isdir(dataset):
        raise FileNotFoundError("The dataset %s is missing!" % (red(dataset)))
```

Ok! We can proceed.

Splitting into train and test

Let's say we leave 60% to learning and 40% to testing. Let's proceed to split the dataset in two:

In [16]:

```
learning_percentage = 0.6
```

First we check if the dataset is already split (this might be a re-run):

In [17]:

```
def is_already_split(root):
    """Return a bool indicating if the dataset has already been split."""
    split_warns = ["learning", "testing"]
    for sub_dir in os.listdir(root):
        for split_warn in split_warns:
            if split_warn in sub_dir:
                return True
    return False
```

Then we split the dataset's books as follows:

Since we want the zipfs that the classifier will use to do the classification built on a proportioned dataset, we pick the percentage of books put aside for learning from the minimum number of books for class in the dataset.

In [18]:

```
def split_books(root, percentage):
    """Split the dataset into learning and testing."""
    min_books = math.inf
    for book_class in get_dirs(root):
        books = get_books(book_class)
        min_books = min(min_books, len(books))
    for book_class in get_dirs(root):
        books = get_books(book_class)
        random.seed(42)  # for reproducibility
        random.shuffle(books)  # Shuffling books
        n = int(min_books * percentage)
        learning_set, testing_set = books[:n], books[
            n:]  # splitting books into the two partitions
        # Moving into respective folders
        [
            shutil.copytree(book,
                            "%s/learning/%s" % (root, book[len(root) + 1:]))
            for book in learning_set
        ]
        [
            shutil.copytree(book,
                            "%s/testing/%s" % (root, book[len(root) + 1:]))
```

```
            for book in testing_set
        ]
```

Here we actually run the two functions:

In [19]:

```
for dataset in datasets:
    if is_already_split(dataset):
        print("I believe I've already split the dataset %s!" % (b(dataset)))
    else:
        split_books(dataset, learning_percentage)
```

```
I believe I've already split the dataset authors!
I believe I've already split the dataset periods!
I believe I've already split the dataset recipes!
```

The metrics

Since distributions hold the following properties:

$$q_i > 0 \quad \forall i \in Q, \qquad \sum_{i \in Q} q_i = 1$$

we will use metrics that will exploit this properties.

These metrics must wither have computational complexity $O(\min(n, m))$ (where n and m are respectively the cardinality of distributions P and Q) or be defined only on the intersection of the distributions, for being practically usable (other than for other reasons shown in the `Root of errors` section).

Informations on the metrics used are below:

Kullback Leibler Divergence

$$D_{KL}(P, Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

The KL divergece is defined for all events in a set $P, Q \subseteq X$.

This forces to define the KL for zipfs only on the subset of the events that are shared beetween the two distributions: $X = P \cap Q$.

This ignores all the information about non-sharec events and it is solved via the Jensen Shannon divergence.

Jensen Shannon Divergence

$$JSD(P, Q) = \frac{1}{2} D_{KL}(P, M) + \frac{1}{2} D_{KL}(Q, M) \qquad M = \frac{1}{2}(P + Q)$$

The JS divergence is defined for every event in a set $X = P \cup Q$, it is **symmetric** and has **always a finite value**.

Getting the current implementation

The current implementation works as follows:

Starting from the extended formulation:

$$m_i = \frac{1}{2}(p_i + q_i), \quad p_i = \begin{cases} p_i & i \in P \\ 0 & otherwise \end{cases}, \quad q_i = \begin{cases} q_i & i \in Q \\ 0 & otherwise \end{cases}$$

$$JSD(P, Q) = \frac{1}{2}\sum_{i \in P} p_i \log\frac{p_i}{m_i} + \frac{1}{2}\sum_{j \in Q} q_j \log\frac{q_j}{m_j}$$

Replacing in the formulation m_i:

$$JSD(P, Q) = \frac{1}{2}\sum_{i \in P} p_i \log\frac{p_i}{\frac{1}{2}(p_i + q_i)} + \frac{1}{2}\sum_{j \in Q} q_j \log\frac{q_j}{\frac{1}{2}(p_j + q_j)}$$

Splitting the sums in 3 distinc sets: $S_1 = i \in P \setminus P \cap Q$, $S_2 = i \in P \cap Q$ and $S_3 = i \in Q \setminus P \cap Q$.

$$JSD(P, Q) = JSD_{S_1}(P, Q) + JSD_{S_2}(P, Q) + JSD_{S_3}(P, Q)$$

$$\begin{aligned}
JSD_{S_1}(P, Q) &= \frac{1}{2}\sum_{i \in P \setminus P \cap Q} p_i \log\frac{p_i}{\frac{1}{2}(p_i + q_i)} + \frac{1}{2}\sum_{j \in P \setminus P \cap Q} q_j \log\frac{q_j}{\frac{1}{2}(p_j + q_j)} \\
&= \frac{1}{2}\sum_{i \in P \setminus P \cap Q} p_i \log\frac{p_i}{\frac{1}{2}(p_i + q_i)} \\
&= \frac{1}{2}\sum_{i \in P \setminus P \cap Q} p_i \log\frac{p_i}{\frac{1}{2}p_i} \\
&= \frac{1}{2}\sum_{i \in P \setminus P \cap Q} p_i \log\frac{1}{\frac{1}{2}} \\
&= \frac{1}{2}\sum_{i \in P \setminus P \cap Q} p_i \log 2 \\
&= \frac{1}{2}\log 2 \sum_{i \in P \setminus P \cap Q} p_i
\end{aligned}$$

$$\begin{aligned}
JSD_{S_2}(P, Q) &= \frac{1}{2}\sum_{i \in P \cap Q} p_i \log\frac{p_i}{\frac{1}{2}(p_i + q_i)} + \frac{1}{2}\sum_{j \in P \cap Q} q_j \log\frac{q_j}{\frac{1}{2}(p_j + q_j)} \\
&= \frac{1}{2}\sum_{i \in P \cap Q} p_i \log\frac{p_i}{\frac{1}{2}(p_i + q_i)} + q_i \log\frac{q_i}{\frac{1}{2}(p_i + q_i)} \\
&= \frac{1}{2}\sum_{i \in P \cap Q} p_i \log\frac{2p_i}{p_i + q_i} + q_i \log\frac{2q_i}{p_i + q_i}
\end{aligned}$$

$$JSD_{S_3}(P, Q) = \frac{1}{2}\log 2 \sum_{j \in Q \setminus P \cap Q} q_j$$

Summing JSD_{S_1} and JSD_{S_3} we can obtain:

$$JSD_{S_1} + JSD_{S_3} = \frac{1}{2}\log 2\left(\sum_{i \in P \setminus P \cap Q} p_i + \sum_{j \in Q \setminus P \cap Q} q_j\right)$$

In particular, if $\sum_{j \in Q}^{m} q_j = 1$ and $\sum_{i \in P}^{n} p_i = 1$, we can write:

$$\begin{aligned} JSD_{S_1} + JSD_{S_3} &= \frac{1}{2}\log 2\left(2 - \sum_{i \in P \cap Q} p_i - \sum_{j \in P \cap Q} q_j\right) \\ &= \frac{1}{2}\log 2\left(2 - \sum_{i \in P \cap Q} p_i + q_j\right) \end{aligned}$$

Putting all togheter we obtain:

$$JSD(P, Q) = \frac{1}{2}\left[\sum_{i \in P \cap Q}\left(p_i \log\frac{2p_i}{p_i + q_i} + q_i \log\frac{2q_i}{p_i + q_i}\right) + \log 2\left(2 - \sum_{i \in P \cap Q} p_i + q_j\right)\right]$$

What's marvelous about this semplification is that the computational complexity decrease from a naive literal interpretation of the initial formula of $O(n + m)$ to $O(\min(n, m))$ simply choosing to iterate over whichever of the two distributions holds less events.

The process is nearly identical for all other metrics shown below:

Hellinger

Given two distributions $P, Q \subseteq X$, the **Hellinger distance** is defined as follows:

$$H(P, Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_{i \in X}\left(\sqrt{p_i} - \sqrt{q_i}\right)^2}$$

Achieving the current implementation

Given $\sum_{i \in P}^{n} p_i = 1$ and $\sum_{i \in Q}^{m} q_i = 1$, we can proceed by separating the sum inside the squared root into three distinct partitions of X: $S_1 = i \in P \setminus P \cap Q$, $S_2 = i \in P \cap Q$ and $S_3 = i \in Q \setminus P \cap Q$.

$$H(P, Q) = \frac{1}{\sqrt{2}}\sqrt{H_{S_1}(P, Q) + H_{S_2}(P, Q) + H_{S_3}(P, Q)}$$

Recalling the definitions of p_i, q_i:

$$p_i = \begin{cases} p_i & i \in P \\ 0 & otherwise \end{cases}, \quad q_i = \begin{cases} q_i & i \in Q \\ 0 & otherwise \end{cases}$$

We begin from $H_{S_1}(P,Q)$:

$$
\begin{aligned}
H_{S_1}(P,Q) &= \sum_{i \in P \setminus P \cap Q} \left(\sqrt{p_i} - \sqrt{q_i}\right)^2 \\
&= \sum_{i \in P \setminus P \cap Q} \left(\sqrt{p_i}\right)^2 \\
&= \sum_{i \in P \setminus P \cap Q} p_i \\
&= 1 - \sum_{i \in P \cap Q} p_i
\end{aligned}
$$

We solve $H_{S_2}(P,Q)$:

$$
H_{S_2}(P,Q) = \sum_{i \in P \cap Q} \left(\sqrt{p_i} - \sqrt{q_i}\right)^2
$$

Now we solve $H_{S_3}(P,Q)$:

$$
\begin{aligned}
H_{S_3}(P,Q) &= \sum_{i \in Q \setminus P \cap Q} q_i \\
&= 1 - \sum_{i \in P \cap Q} q_i
\end{aligned}
$$

Now, putting it all togheter we have:

$$
\begin{aligned}
H_{S_1}(P,Q) + H_{S_2}(P,Q) + H_{S_3}(P,Q) &= 2 + \sum_{i \in P \cap Q} \left[\left(\sqrt{p_i} - \sqrt{q_i}\right)^2 - p_i - q_i\right] \\
&= 2 + \sum_{i \in P \cap Q} - 2\sqrt{p_i q_i} \\
&= 2\left(1 - \sum_{i \in P \cap Q} \sqrt{p_i q_i}\right)
\end{aligned}
$$

So the Hellinger distance is redefined as:

$$
\begin{aligned}
H(P,Q) &= \frac{1}{\sqrt{2}} \sqrt{2\left(1 - \sum_{i \in P \cap Q} \sqrt{p_i q_i}\right)} \\
&= \sqrt{1 - \sum_{i \in P \cap Q} \sqrt{p_i q_i}}
\end{aligned}
$$

In [20]:

```
metrics = [
    jensen_shannon,
    normal_total_variation,
```

```
    intersection_total_variation,
    kullback_leibler,
    intersection_squared_hellinger,
    hellinger,
    squared_hellinger
]
```

In [21]:

```
for metric in metrics:
    print_function(metric)
```

```
def jensen_shannon(a: dict, b: dict)->float:
    """Return the jensen shannon divergence beetween a and b."""
    total = 0
    delta = 0
    big, small = sort(a, b)

    big_get = big.__getitem__

    for key, small_value in small.items():
        try:
            big_value = big_get(key)
            if big_value:
                denominator = (big_value + small_value) / 2
                total += small_value * log(small_value / denominator
) + \
                    big_value * log(big_value / denominator)
                delta += big_value + small_value
        except KeyError:
            pass

    total += (2 - delta) * log(2)
    return total / 2

def normal_total_variation(a: dict, b: dict) -> float:
    """Determine the Normalized Total Variation distance."""
    big, small = sort(a, b)
    big_get = big.__getitem__
    total = 2
    for k, small_value in small.items():
        try:
            big_value = big_get(k)
            if big_value:
                total += abs(big_value - small_value) - big_value -
small_value
        except KeyError:
            pass
    return total / 2

def intersection_total_variation(a: dict, b: dict, overlap: bool=Fal
se)->float:
    """Return the total distance beetween the intersection of a and
 b."""
    return intersection_nth_variation(a, b, 1, overlap)
```

```python
def kullback_leibler(a: dict, b: dict) -> float:
    """Determine the Kullback Leibler divergence."""
    total = 0
    big, small = sort(a, b)
    big_get = big.__getitem__
    for key, small_value in a.items():
        try:
            big_value = big_get(key)
            if big_value:
                total += small_value * log(small_value / big_value)
        except KeyError:
            pass

    return total

def intersection_squared_hellinger(a: dict, b: dict) -> float:
    """Determine the Intersection Squared Hellinger distance."""
    total = 0
    big, small = sort(a, b)
    big_get = big.__getitem__
    for key, small_value in small.items():
        try:
            total += (sqrt(small_value) - sqrt(big_get(key)))**2
        except KeyError:
            pass
    return total

def hellinger(a: dict, b: dict) -> float:
    """Determine the Hellinger distance."""
    try:
        v = squared_hellinger(a, b)
        return sqrt(v)
    except ValueError as e:
        if isclose(v, 0, abs_tol=1e-15):
            return 0
        raise e

def squared_hellinger(a: dict, b: dict) -> float:
    """Determine the Squared Hellinger distance."""
    total = 1
    big, small = sort(a, b)
    big_get = big.__getitem__
    for key, small_value in small.items():
        try:
            total -= sqrt(small_value * big_get(key))
        except KeyError:
            pass
    return total
```

The options

We will use the following options for learning and testing. More informations about options' customizations is available here. In this test we use simply the default settings (a plain zipf) with no stop word removal or cardinality removal.

A couple examples

Possible options could be to remove english or italian stop words (the stop words list is in the package zipf):

```
{
  "remove_stop_words": false, # Removes stop words
  "stop_words": "it" # Removes italian stop words
}
```

```
{
  "remove_stop_words": false, # Removes stop words
  "stop_words": "en" # Removes english stop words
}
```

Or to remove words that appear less than a given time:

```
{
  "minimum_count": 1, # Removes words that appear less than 'minimum_coun
t'
}
```

Chaining options are available but the current implementation uses too much memory to be of practically usable.

In [22]:

```
options = {}
```

Creating the Zipfs

We will now convert all the chapters in the dataset into the respective zipf for each option.

In [23]:

```
def create_zipfs(paths, factory, test_path):
    for data_path in tqdm(paths, unit=' zipf'):
        path = "%s/%s.json" % (test_path, '/'.join(data_path.split('/')[1:]))
        # If the zipf already exists we skip it
        if os.path.exists(path):
            continue
        path_dirs = '/'.join(path.split('/')[:-1])
        zipf = factory.run(data_path, ['txt'])
        if not zipf.is_empty():
            if not os.path.exists(path_dirs):
                os.makedirs(path_dirs)
            zipf.save(path)
```

We define the paths for zipfs and their sources:

In [24]:

```python
def get_build_paths(dataset):
    """Return a triple with the build paths for given dataset."""
    learning_path = "%s/learning" % dataset
    testing_path = "%s/testing" % dataset
    zipfs_path = '%s/zipfs' % dataset

    print(
        "I will build learning zipfs from %s,\ntesting zipfs from %s\nand save t
hem in %s\n"
        % (b(learning_path), b(testing_path), b(zipfs_path)))
    return learning_path, testing_path, zipfs_path
```

First we create the learning zipfs:

In [25]:

```python
def build_learning_zipfs(path, zipfs_path):
    """Build zipfs from txt files at given path."""
    print("Creating learning zipfs in %s" % (b(path)))
    book_classes = get_dirs(path)
    print("Some of the paths I'm converting are:")
    random.seed(42)  # For reproducibility
    random.shuffle(book_classes)
    shown = []
    for book in book_classes[:10]:
        shown.append((book, ''))
    display(
        HTML(
            tabulate.tabulate(shown, ["Learning data paths", ''],
                              tablefmt='html')))
    create_zipfs(book_classes, factory, zipfs_path)
```

And then the testing zipfs:

In [26]:

```python
def build_testing_zipfs(path, zipfs_path):
    """Build zipfs from txt files at given path."""
    print("Creating testing zipfs in %s" % (b(path)))
    books = get_books(path)
    random.seed(42)  # For reproducibility
    random.shuffle(books)
    shown = []
    for book in books[:10]:
        shown.append((book, ''))
    display(
        HTML(
            tabulate.tabulate(
                shown, ["Testing data paths", ''], tablefmt='html')))
    create_zipfs(books, factory, zipfs_path)
```

We create a factory for creating the zipfs objects from files with the options defined above. More informations about factory customization and other possible factories is available here.

In [27]:

```python
factory = ZipfFromDir(options=options)
print("Created a factory with options %s" % (factory))
```

```
Created a factory with options {
  "remove_stop_words": false,
  "stop_words": "it",
  "minimum_count": 0,
  "chain_min_len": 1,
  "chain_max_len": 1,
  "chaining_character": " ",
  "sort": false
}
```

Wake up zipfs factory daemons:

In [28]:

```
factory.start_processes()
```

Actually creating the zipfs:

In [29]:

```
for dataset in datasets:
    print("Building dataset %s" % (b(dataset)))
    learning_path, testing_path, zipfs_path = get_build_paths(dataset)
    build_learning_zipfs(learning_path, zipfs_path)
    build_testing_zipfs(testing_path, zipfs_path)
    print(gray('=' * frame_number))
```

```
Building dataset authors
I will build learning zipfs from authors/learning,
testing zipfs from authors/testing
and save them in authors/zipfs

Creating learning zipfs in authors/learning
Some of the paths I'm converting are:
```

Learning data paths
authors/learning/twain
authors/learning/dh_lawrence
authors/learning/wilde

```
Creating testing zipfs in authors/testing
```

Testing data paths
authors/testing/twain/3275
authors/testing/twain/320
authors/testing/wilde/florentine-tragedy
authors/testing/dh_lawrence/4483
authors/testing/wilde/2252
authors/testing/twain/3297
authors/testing/wilde/2305
authors/testing/dh_lawrence/fantasia-of-unconscious

Testing data paths
authors/testing/wilde/2317
authors/testing/twain/3259

```
==============================
Building dataset **periods**
I will build learning zipfs from **periods/learning**,
testing zipfs from **periods/testing**
and save them in **periods/zipfs**

Creating learning zipfs in **periods/learning**
Some of the paths I'm converting are:
```

Learning data paths
periods/learning/romanticism
periods/learning/realism
periods/learning/naturalism
periods/learning/modernism

```
Creating testing zipfs in **periods/testing**
```

Testing data paths
periods/testing/romanticism/579
periods/testing/modernism/3452
periods/testing/romanticism/550
periods/testing/romanticism/2124
periods/testing/romanticism/4545
periods/testing/romanticism/491
periods/testing/modernism/3484
periods/testing/romanticism/143
periods/testing/modernism/blanco-posnet
periods/testing/realism/indian-summer

```
==============================
Building dataset **recipes**
I will build learning zipfs from **recipes/learning**,
testing zipfs from **recipes/testing**
and save them in **recipes/zipfs**

Creating learning zipfs in **recipes/learning**
Some of the paths I'm converting are:
```

Learning data paths
recipes/learning/recipes
recipes/learning/non_recipes

```
Creating testing zipfs in recipes/testing
```

Testing data paths
recipes/testing/non_recipes/409d884a6896b8673a0643cb615a7d4b
recipes/testing/non_recipes/7e36a2f244dbce34106e0b1c9b9b41ca
recipes/testing/recipes/955d93252cccff4b4d7943b8e678e367
recipes/testing/recipes/63ec182e7c66babebcd4a2cc487b098b
recipes/testing/non_recipes/b456e579f6cd6939e5013d250c7fe0ab
recipes/testing/non_recipes/71eb580781257553cd6c850c8144492d
recipes/testing/non_recipes/efdce7aa0c2c0c3c61f96ccc77e0f029
recipes/testing/non_recipes/9edf96e7b7f8251c85c14f64e9df15d7
recipes/testing/non_recipes/fd68ae7002749fe0e63a581fead0ff35
recipes/testing/non_recipes/9598041f7ffb1393f1a02f3f81d127d8

```
==============================
```

Slaying daemons:

In [30]:

```
factory.close_processes()
```

Creating the Classifier

Now we have rendered the learning. Let's run some tests!

The classifier works as follows:

Given a function $z(d): W^u -> [0,1]^v, \quad u \leq v$ a function to convert a document into a zipf where d is a list of words and W is the domain of possible words, a metric $m(P,Q): [0,1]^n \times [0,1]^m -> \mathrm{R}$, a learning set L of k tuples (l_i, Z_i), where l_i is the label of the set of zipfs Z_i, we proceed to classify a given document d via two steps:

1. Convert the document d to zipf: $z_d = z(d)$
2. Predicted label is $l^* = \text{argmin}_{l_i} \left\{ (l_i, Z_i) : \frac{1}{\#\{Z_i\}} \sum_{z \in Z_i} m(z_d, z) \right\}$, where $\#\{Z_i\}$ is the cardinality of Z_i.

In [31]:

```
def get_classifier_paths(dataset):
    """Return paths for classifier, given a dataset."""
    zipfs_path = get_build_paths(dataset)[2]
    learning_zipfs_path = "%s/learning" % zipfs_path
    testing_zipfs_path = "%s/testing" % zipfs_path
    return learning_zipfs_path, testing_zipfs_path
```

In [32]:

```
def load_zipfs(classifier, path):
    """Load zipfs from given path into given classifier."""
    print("Loading zipfs from %s" % (b(path)))
    loaded = []
    for zipf in tqdm(get_zipfs(path), unit='zipf'):
        book_class = zipf.split('/')[-1].split('.')[0]
        args = zipf, book_class
        loaded.append(args)
        classifier.add_zipf(*args)

    random.seed(42)
    random.shuffle(loaded)
    display(
        HTML(
            tabulate.tabulate(loaded[:10], ["Path", "Class"],
                              tablefmt='html')))
```

In [33]:

```
def load_test_couples(path):
    """Return list of zipfs from given path."""
    print("Loading tests from %s" % (b(path)))
    test_couples = []
    for zipf in tqdm(get_zipfs(path)):
        book_class = zipf.split('/')[-2]
        args = zipf, book_class
        test_couples.append(args)

    random.seed(42)
    random.shuffle(test_couples)
    display(
        HTML(
            tabulate.tabulate(
                test_couples[:10], ["Path", "Class"], tablefmt='html')))
    return test_couples
```

In [34]:

```
def metrics_test(classifier, test_couples):
    """Run test on all metrics usable on zipfs."""
    global metrics
    for metric in metrics:
        results = classifier.test(test_couples, metric)
        success(results, metric)
```

First we create the classifier with the options set above:

In [35]:

```
classifier = ZipfClassifier(options)
print("We're using a classifier with options %s" % classifier)
```

```
We're using a classifier with options {
    "sort": false
}
```

In [36]:

```
for dataset in datasets:
    print("Testing dataset %s" % (b(dataset)))
    learning_zipfs_path, testing_zipfs_path = get_classifier_paths(dataset)
    print(gray('-' * frame_number))
    load_zipfs(classifier, learning_zipfs_path)
    print(gray('-' * frame_number))
    test_couples = load_test_couples(testing_zipfs_path)
    print(gray('-' * frame_number))
    metrics_test(classifier, test_couples)
    classifier.clear()
    print(gray('=' * frame_number))
```

```
Testing dataset authors
I will build learning zipfs from authors/learning,
testing zipfs from authors/testing
and save them in authors/zipfs

------------------------------
Loading zipfs from authors/zipfs/learning
```

Path	Class
authors/zipfs/learning/dh_lawrence.json	dh_lawrence
authors/zipfs/learning/twain.json	twain
authors/zipfs/learning/wilde.json	wilde

```
------------------------------
Loading tests from authors/zipfs/testing
```

Path	Class
authors/zipfs/testing/twain/double-barrelled.json	twain
authors/zipfs/testing/twain/3294.json	twain
authors/zipfs/testing/wilde/2280.json	wilde
authors/zipfs/testing/dh_lawrence/3484.json	dh_lawrence
authors/zipfs/testing/wilde/2299.json	wilde
authors/zipfs/testing/twain/3277.json	twain
authors/zipfs/testing/wilde/2318.json	wilde
authors/zipfs/testing/dh_lawrence/3487.json	dh_lawrence
authors/zipfs/testing/wilde/2288.json	wilde
authors/zipfs/testing/twain/323.json	twain

```
------------------------------
Success with metric jensen_shannon: 79.38%
```

Info	Values
success	127
failures	32

Info	Values
unclassified	1
mean_delta	0.0180276
Mistook wilde for dh_lawrence	16
Mistook wilde for twain	16

```
Success with metric normal_total_variation: 73.75%
```

Info	Values
success	118
failures	42
unclassified	0
mean_delta	0.0265837
Mistook wilde for dh_lawrence	15
Mistook wilde for twain	21
Mistook dh_lawrence for twain	5
Mistook twain for wilde	1

```
Success with metric intersection_total_variation: 78.75%
```

Info	Values
success	126
failures	34
unclassified	0
mean_delta	0.0366234
Mistook twain for wilde	3
Mistook wilde for dh_lawrence	10
Mistook dh_lawrence for wilde	4
Mistook wilde for twain	14
Mistook dh_lawrence for twain	2
Mistook twain for dh_lawrence	1

```
Success with metric kullback_leibler: 60.0%
```

Info	Values
success	96
failures	64
unclassified	0
mean_delta	0.123656
Mistook dh_lawrence for wilde	26

Info	Values
Mistook twain for wilde	20
Mistook wilde for dh_lawrence	13
Mistook wilde for twain	1
Mistook twain for dh_lawrence	4

Success with metric intersection_squared_hellinger: **87.5%**

Info	Values
success	140
failures	20
unclassified	0
mean_delta	0.0367975
Mistook wilde for dh_lawrence	8
Mistook wilde for twain	10
Mistook dh_lawrence for twain	1
Mistook dh_lawrence for wilde	1

Success with metric hellinger: **81.25%**

Info	Values
success	130
failures	30
unclassified	0
mean_delta	0.0217585
Mistook wilde for dh_lawrence	15
Mistook wilde for twain	15

Success with metric squared_hellinger: **81.25%**

Info	Values
success	130
failures	30
unclassified	0
mean_delta	0.0259997
Mistook wilde for dh_lawrence	15
Mistook wilde for twain	15

==============================

Testing dataset **periods**
I will build learning zipfs from **periods/learning,**
testing zipfs from **periods/testing**
and save them in **periods/zipfs**

Loading zipfs from **periods/zipfs/learning**

Path	Class
periods/zipfs/learning/romanticism.json	romanticism
periods/zipfs/learning/realism.json	realism
periods/zipfs/learning/naturalism.json	naturalism
periods/zipfs/learning/modernism.json	modernism

Loading tests from **periods/zipfs/testing**

Path	Class
periods/zipfs/testing/romanticism/680.json	romanticism
periods/zipfs/testing/modernism/sea-and-sardinia.json	modernism
periods/zipfs/testing/romanticism/158.json	romanticism
periods/zipfs/testing/romanticism/fugitive-pieces.json	romanticism
periods/zipfs/testing/romanticism/3882.json	romanticism
periods/zipfs/testing/romanticism/129.json	romanticism
periods/zipfs/testing/modernism/misalliance.json	modernism
periods/zipfs/testing/romanticism/684.json	romanticism
periods/zipfs/testing/modernism/in-the-cage.json	modernism
periods/zipfs/testing/realism/4137.json	realism

Success with metric jensen_shannon: **63.77%**

Info	Values
success	551
failures	313
unclassified	0
mean_delta	0.00737391
Mistook modernism for naturalism	16
Mistook romanticism for realism	137
Mistook modernism for realism	31
Mistook realism for romanticism	25
Mistook romanticism for naturalism	11
Mistook naturalism for realism	27
Mistook realism for naturalism	5
Mistook naturalism for modernism	16

Info	Values
Mistook realism for modernism	10
Mistook modernism for romanticism	23
Mistook naturalism for romanticism	3
Mistook romanticism for modernism	9

```
Success with metric normal_total_variation: 58.8%
```

Info	Values
success	508
failures	356
unclassified	0
mean_delta	0.0114856
Mistook modernism for naturalism	20
Mistook modernism for realism	27
Mistook romanticism for realism	133
Mistook realism for romanticism	26
Mistook romanticism for modernism	37
Mistook romanticism for naturalism	24
Mistook naturalism for realism	17
Mistook realism for naturalism	8
Mistook naturalism for modernism	24
Mistook realism for modernism	12
Mistook naturalism for romanticism	3
Mistook modernism for romanticism	25

```
Success with metric intersection_total_variation: 59.49%
```

Info	Values
success	514
failures	350
unclassified	0
mean_delta	0.0246552
Mistook romanticism for realism	40
Mistook modernism for naturalism	30
Mistook romanticism for modernism	85
Mistook realism for modernism	23
Mistook realism for romanticism	63
Mistook romanticism for naturalism	21

Info	Values
Mistook modernism for realism	8
Mistook realism for naturalism	17
Mistook modernism for romanticism	33
Mistook naturalism for romanticism	8
Mistook naturalism for modernism	20
Mistook naturalism for realism	2

`Success with metric kullback_leibler: 63.54%`

Info	Values
success	549
failures	315
unclassified	0
mean_delta	0.116207
Mistook modernism for naturalism	25
Mistook romanticism for naturalism	15
Mistook modernism for realism	33
Mistook realism for romanticism	55
Mistook realism for naturalism	10
Mistook romanticism for realism	40
Mistook modernism for romanticism	41
Mistook naturalism for realism	34
Mistook realism for modernism	26
Mistook naturalism for romanticism	15
Mistook naturalism for modernism	14
Mistook romanticism for modernism	7

`Success with metric intersection_squared_hellinger: 79.51%`

Info	Values
success	687
failures	177
unclassified	0
mean_delta	0.0207118
Mistook modernism for naturalism	24
Mistook realism for romanticism	44
Mistook realism for naturalism	10
Mistook modernism for realism	7

Info	Values
Mistook modernism for romanticism	36
Mistook romanticism for naturalism	4
Mistook naturalism for modernism	11
Mistook romanticism for realism	7
Mistook realism for modernism	12
Mistook naturalism for romanticism	9
Mistook romanticism for modernism	9
Mistook naturalism for realism	4

`Success with metric hellinger: 65.39%`

Info	Values
success	565
failures	299
unclassified	0
mean_delta	0.00946594
Mistook modernism for realism	27
Mistook romanticism for realism	137
Mistook realism for romanticism	24
Mistook modernism for naturalism	10
Mistook romanticism for naturalism	6
Mistook naturalism for realism	32
Mistook realism for naturalism	4
Mistook naturalism for modernism	17
Mistook realism for modernism	7
Mistook modernism for romanticism	27
Mistook naturalism for romanticism	4
Mistook romanticism for modernism	4

`Success with metric squared_hellinger: 65.39%`

Info	Values
success	565
failures	299
unclassified	0
mean_delta	0.0107078
Mistook modernism for realism	27

Info	Values
Mistook romanticism for realism	137
Mistook realism for romanticism	24
Mistook modernism for naturalism	10
Mistook romanticism for naturalism	6
Mistook naturalism for realism	32
Mistook realism for naturalism	4
Mistook naturalism for modernism	17
Mistook realism for modernism	7
Mistook modernism for romanticism	27
Mistook naturalism for romanticism	4
Mistook romanticism for modernism	4

```
==============================
Testing dataset recipes
I will build learning zipfs from recipes/learning,
testing zipfs from recipes/testing
and save them in recipes/zipfs

------------------------------
Loading zipfs from recipes/zipfs/learning
```

Path	Class
recipes/zipfs/learning/non_recipes.json	non_recipes
recipes/zipfs/learning/recipes.json	recipes

```
------------------------------
Loading tests from recipes/zipfs/testing
```

Path	Class
recipes/zipfs/testing/non_recipes/3d42a56c681d56a73985877ba3c8bd6f.json	non_recipes
recipes/zipfs/testing/non_recipes/7c3d10eb81a2660e81cff3c3b41a951e.json	non_recipes
recipes/zipfs/testing/recipes/e2cd675ec121020b224e5118b5ae4d1b.json	recipes
recipes/zipfs/testing/recipes/2a90bc4cd74f6bbc578c9c5d8fda741f.json	recipes
recipes/zipfs/testing/non_recipes/bfbc1b5eee1d423942b8b0b55f82bbff.json	non_recipes
recipes/zipfs/testing/non_recipes/6c41a532214cc6dfdbcc9b3bb5be0630.json	non_recipes
recipes/zipfs/testing/non_recipes/c31a242182356ea14acf080f628b03cc.json	non_recipes
recipes/zipfs/testing/non_recipes/bbecd5cafac80b5441765704a18db2c3.json	non_recipes
recipes/zipfs/testing/non_recipes/3a2b4e9cb1130e371f12e48525e2e190.json	non_recipes
recipes/zipfs/testing/non_recipes/38928442e96835f1897ab5c1cb4276a3.json	non_recipes

```
------------------------------
```

Success with metric jensen_shannon: **85.71%**

Info	Values
success	6183
failures	1030
unclassified	1
mean_delta	0.0484755
Mistook non_recipes for recipes	1030

Success with metric normal_total_variation: **72.04%**

Info	Values
success	5197
failures	2016
unclassified	1
mean_delta	0.0538705
Mistook non_recipes for recipes	2016

Success with metric intersection_total_variation: **93.64%**

Info	Values
success	6755
failures	459
unclassified	0
mean_delta	0.0799341
Mistook non_recipes for recipes	458
Mistook recipes for non_recipes	1

Success with metric kullback_leibler: **28.64%**

Info	Values
success	2066
failures	5148
unclassified	0
mean_delta	0.848787
Mistook non_recipes for recipes	5148

Success with metric intersection_squared_hellinger: **99.57%**

Info	Values
success	7183
failures	31
unclassified	0

Info	Values
mean_delta	0.148349
Mistook non_recipes for recipes	31

```
Success with metric hellinger: 92.47%
```

Info	Values
success	6671
failures	541
unclassified	2
mean_delta	0.0540858
Mistook non_recipes for recipes	541

```
Success with metric squared_hellinger: 92.49%
```

Info	Values
success	6672
failures	541
unclassified	1
mean_delta	0.0821632
Mistook non_recipes for recipes	541

```
==============================
```

Root of errors

Here follows a list of some of the possible cause of errors I have diagnosed in the dataset formulation:

Cardinality of difference of sets

The difference of the two sets has an important effect on the good result of the classification: if $\#\{P \setminus P \cap Q\} >> \#\{Q \setminus P \cap Q\}$ the metric should include only the intersection. It is for this reason that the `intersection_squared_hellinger` metric works best generally, but expecially in these situation such in the case of datasets **periods.**

Small texts

When a text is significantly smaller than the average element in the learning set it will only be marked as a false positive or negative. In these datasets I have removed elements with less than 200 characters for this reason, since they do not offer enough informations for a significant classification.

Conclusions

The classification method proposed, expecially using the `intersection_squared_hellinger` is extremely fast: in average it converts to zipf an average recepy and test it in **1.06 ms ± 50.3 µs**. It also is, in the case of web articles in particular, in average, correct and consistent: it could be used as a fast catalogation tecnique for focused web crawlers as a part of the filters to remove unwanted content. Combinations of the distances proposed might bring an higher success rate.

Test computer specifications

The computer on which the metrics where timed had the following specifications:

Computer specifications	
Model Name	MacBook Pro
Processor Name	Intel Core i7
Processor Speed	2.3 GHz
Number of Processors	1
Total Number of Cores	4
L2 Cache (per Core)	256 KB
L3 Cache	6 MB
Memory	16 GB

Future works

In the near future, I'll develop the classifier using a learning algoritms to determine which combinations of distances achieves the best success rate. Also, I'll be trying to use this classifier as a way to power an autonomous crawler starting from the current implementation of an other project of mine TinyCrawler.