# WikiSearch

Mami Ciro Antonio
January 14th 2021

UNIVERSITÀ DEGLI STUDI DI TRIESTE

# What WikiSearch does

WikiSearch is a tool that aims to emulate a search engine. The user insert a query through a command-line interface and the most relevants Wikipedia pages are retrieved from the simple Wikipedia dump of April 2007

```
SEARCH (digit exit to close):radius
Mathematics
Number
Set
Function
Integer
Multiplication
English_language
Science
Arithmetic
Theorem
SEARCH (digit exit to close):
```

# What's under WikiSearch

The most relevant pages are retrieved according to the Topic-Sensitive PageRank developed at Stanford University.

# PageRank

PageRank is the algorithm originally used by Google in ranking the pages in its results.

Developed in 1996 by Sergej Brin and Larry Page, the main idea is to assign a static score to a page, which is independent from the query and depends on the number and the quality of incoming links.

Formally, this is achieved exploiting Markov chain.

The first step consist in extracting the transition matrix $R$ from the data.
$R$ is defined as follow:

$$R_{i,j} = \begin{cases} 0, & \text{if } i \nrightarrow j \\ \frac{1}{O[i]}, & \text{if } i \rightarrow j \end{cases} \quad (1)$$

Where $O[i]$ is the total number of outgoing links for page $i$.

Now, thanks to property of Markov chains, we can find the limit distribution of the importance (rank) of each page with the iterative equation:

$$\vec{Rank}_t = \vec{Rank}_{t-1} \cdot (\alpha \vec{1}^T \vec{J} + (1 - \alpha) R) \tag{2}$$

Where $\vec{J}$ is called Jump Vector and models the probability of the user randomly going to a new page instead of following the links. This is a very powerful tool in biasing our algorithm.
Equation (2) is computed until convergence.

# Topic-Sensitive PageRank

Published in 2002 this algorithm aims to compute a PageRank vector which depends on the query, answering the question "which are the most relevant pages for this query?".
The algorithm consists in two phases, offline and online.

Offline, a set of PageRank vectors is computed biased on a set of representative topics.
At the same time, a term-frequency dictionary is computed for each topic, based on the terms occurrences in the corpus of the pages.

We can bias PageRank on a specific topic by selecting a set of pages $S$, which are related to the topic, and defining a new Jump Vector $\vec{J}_S$ as:

$$\vec{J_{Si}} = \begin{cases} \frac{1}{|S|}, & \text{if } i \in S \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

At query time, we infer the probability of a specific topic $c_j$ given the query $q$. If the query consists of multiple terms this can be computed as:

$$P(c_j|q) = \frac{P(c_j) \cdot P(q|c_j)}{P(q)} \propto P(c_j) \prod_i P(q_i|c_j) \tag{4}$$

For our purpose we will assume that the topics are uniformly distributed and so $P(cj)$ can be omitted.

Finally the PageRank score $s$ for a page $p$ given a query $q$ is estimated according to:

$$s_{q,p} = \sum_j P(c_j|q) \cdot rank_j, \tag{5}$$

For this project the topics have been selected among the most frequent keywords on the dataset and are:

- Mathematics
- Football
- Film
- Government
- Music
- Book
- Food
- Computer
- Actor
- Animal
- Plant

# Coding Topic-Sensitive PageRank

# Building the Dataset

The script `dataset.py` returns two `.json` files:

- `data.json` which contains a dictionary where the keys are the Wikipedia pages and the values are their outgoing links.

# Building the Dataset

The script `dataset.py` returns two `.json` files:

- `data.json` which contains a dictionary where the keys are the Wikipedia pages and the values are their outgoing links.
- `meta.json` a dictionary where the keys are the Wikipedia pages and the values are their keywords.

In order to obtain those dictionary, the following steps are required:

- Some unuseful pages (e.g. `'index.html'`) are filtered out.

In order to obtain those dictionary, the following steps are required:

- Some unuseful pages (e.g. `'index.html'`) are filtered out.
- The files are passed to an html parser which allows to extract metadata and outgoing links.

In order to obtain those dictionary, the following steps are required:

- Some unuseful pages (e.g. `'index.html'`) are filtered out.
- The files are passed to an html parser which allows to extract metadata and outgoing links.
- The filenames are normalized to be coherent.

In order to obtain those dictionary, the following steps are required:

- Some unuseful pages (e.g. `'index.html'`) are filtered out.
- The files are passed to an html parser which allows to extract metadata and outgoing links.
- The filenames are normalized to be coherent.
- Finally all the outgoing links which points to an external page are filtered out from the dictionary.

# Implementation of the algorithm

The script `pagerank.py` implements the class `PageRank` which computes the classic PageRank algorithm.

The class needs as input a dictionary (i.e. the dict extracted from `data.json`).

The script `corpus.py` contains a simple function `extract_corpus` which takes as input an html file and extracts the content of the page already preprocessed with:

- Tokenization.

The script `corpus.py` contains a simple function `extract_corpus` which takes as input an html file and extracts the content of the page already preprocessed with:

- Tokenization.
- Removal of puctuation.

The script `corpus.py` contains a simple function `extract_corpus` which takes as input an html file and extracts the content of the page already preprocessed with:

- Tokenization.
- Removal of puctuation.
- To lowercase.

The script `corpus.py` contains a simple function `extract_corpus` which takes as input an html file and extracts the content of the page already preprocessed with:

- Tokenization.
- Removal of puctuation.
- To lowercase.
- Removal of stopwords.

`topicPageRank.py` implements the class `TopicRank`, which inherit from the `PageRank` base class and extends it by allowing to compute topic-sensitive PageRank.

- It takes as input in addition to the dictionary with the graph:

`topicPageRank.py` implements the class `TopicRank`, which inherit from the `PageRank` base class and extends it by allowing to compute topic-sensitive PageRank.

- It takes as input in addition to the dictionary with the graph:
  - the dictionary with the metadata of the pages

`topicPageRank.py` implements the class `TopicRank`, which inherit from the `PageRank` base class and extends it by allowing to compute topic-sensitive PageRank.

- It takes as input in addition to the dictionary with the graph:
  - the dictionary with the metadata of the pages
  - the topic which will bias the pagerank

`topicPageRank.py` implements the class `TopicRank`, which inherit from the `PageRank` base class and extends it by allowing to compute topic-sensitive PageRank.

- It takes as input in addition to the dictionary with the graph:
  - the dictionary with the metadata of the pages
  - the topic which will bias the pagerank
  - [Optional] a vector with a precomputed unbias PageRank

- `compute_J()` returns the jump vector based on the topic and for each Wikipedia page that contain the topic in its metadata, a term-frequency dictionary is updated.

- `compute_J()` returns the jump vector based on the topic and for each Wikipedia page that contain the topic in its metadata, a term-frequency dictionary is updated.
- Before saving, the term-frequency dictionary it is normalized, in this way $tf_j[q] = P(q|c_j)$.

- `compute_J()` returns the jump vector based on the topic and for each Wikipedia page that contain the topic in its metadata, a term-frequency dictionary is updated.
- Before saving, the term-frequency dictionary it is normalized, in this way $tf_j[q] = P(q|c_j)$.
- Finally `update_Rank()` computes the PageRank vector biased on a specific topic.

`wikisearch.py` combines all the previous scripts and returns the most relevant pages for a given query according to equation (4). This is done after that, offline, all the topic-sensitive PageRank and the term-frequency dictionary has been computed.

# Limitations and improvements

- The main limitation to WikiSearch is the quality of data, in particular the metadata extracted from the `.html` files.

# Limitations and improvements

- The main limitation to WikiSearch is the quality of data, in particular the metadata extracted from the `.html` files.
- We can increase the accuracy of keywords by exploiting a Neural Network to model the topics for each document.

# Limitations and improvements

- The main limitation to WikiSearch is the quality of data, in particular the metadata extracted from the `.html` files.
- We can increase the accuracy of keywords by exploiting a Neural Network to model the topics for each document.
- We can bias the algorithm to give an higher score to pages which contains the query in the corpus (using a regularization to avoid scam).