Wikipedia Racer Bot

Made by Dmytro Zakharov, Daniil Korenkov, Kvaratskheliia David

July 21, 2022

GitHub repository:
https://github.com/ZamDimon/ML-Wikipedia-Runner

Plan

1. Introduction
2. Algorithm Description
3. Data Generation
   - Set of Features and Outputs
   - Top words
   - Previous Algorithms
   - Six Degrees of Wikipedia
4. Data Filtering
   - Distance Distribution
   - Specifying connection
5. Training model and Accuracy

What is Wikiracing?

Definition

According to Wikipedia, **Wikiracing** is a game which the players race towards the goal of traversing from one Wikipedia page to another using only internal links.

Our goal

Teach a model to find optimal path from given page to the targeted one.

Algorithm Description

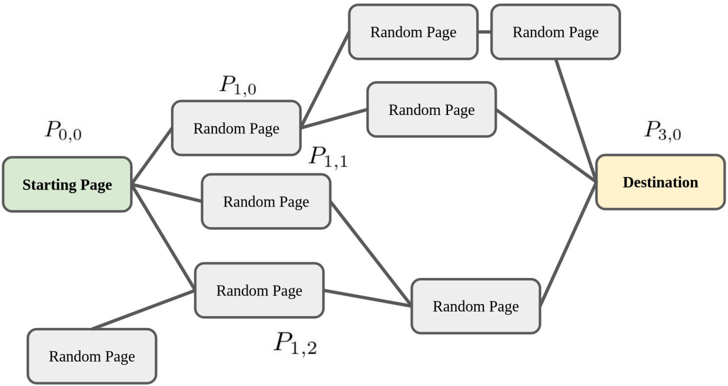


Figure: Environment setup

Algorithm Description

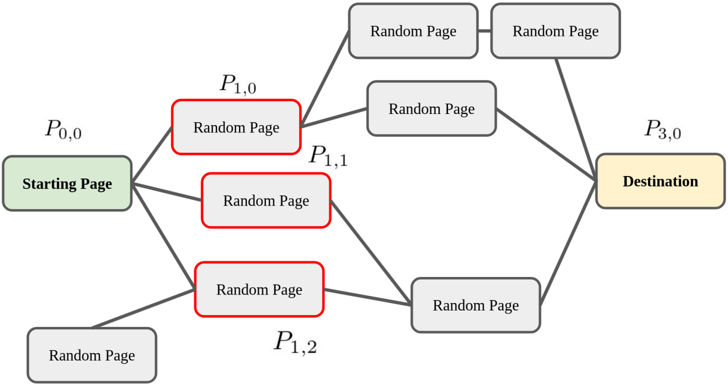


Figure: Forming set $P_{1,j}$

Algorithm Description

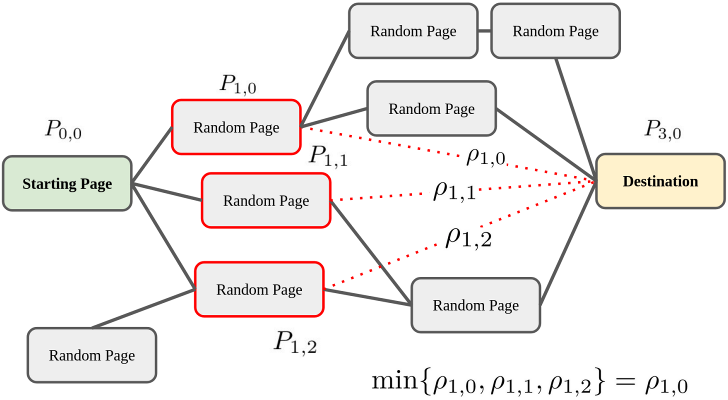


Figure: Evaluating distances $\rho_{1,j} = \rho(P_{1,j}, P_{3,0})$

Algorithm Description

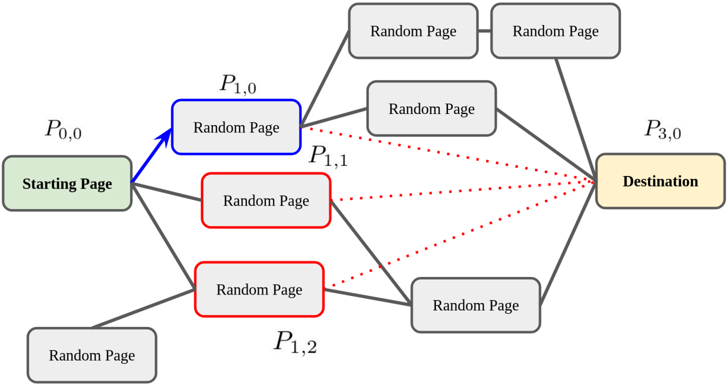


Figure: Choosing page with minimal distance to $P_{3,0}$

Algorithm Description

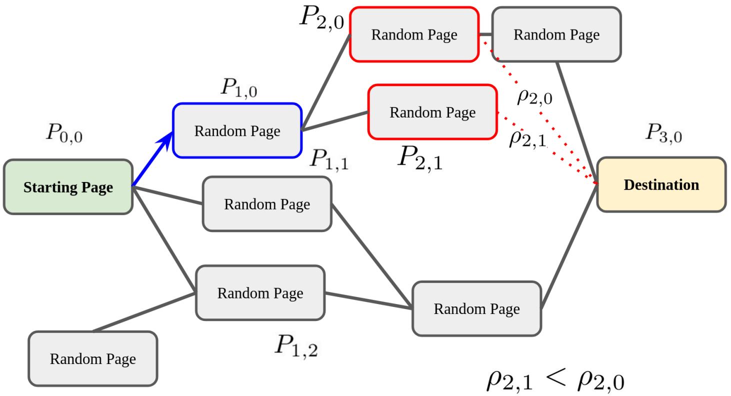


Figure: Forming array $P_{2,k}$ and evaluating corresponding distances $\rho_{2,j} = \rho(P_{2,j}, P_{3,0})$

Algorithm Description

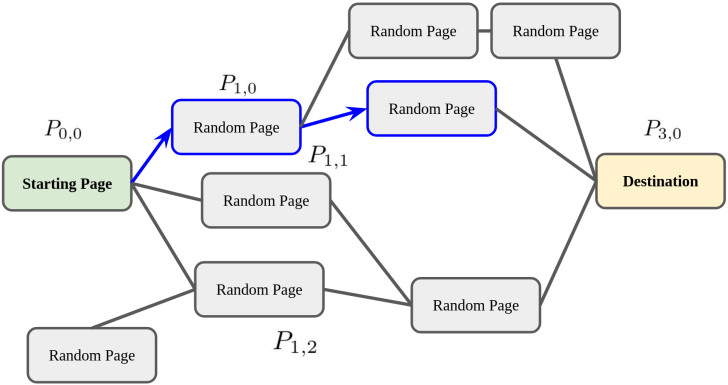


Figure: Choosing page with minimal distance to $P_{3,0}$

Algorithm Description

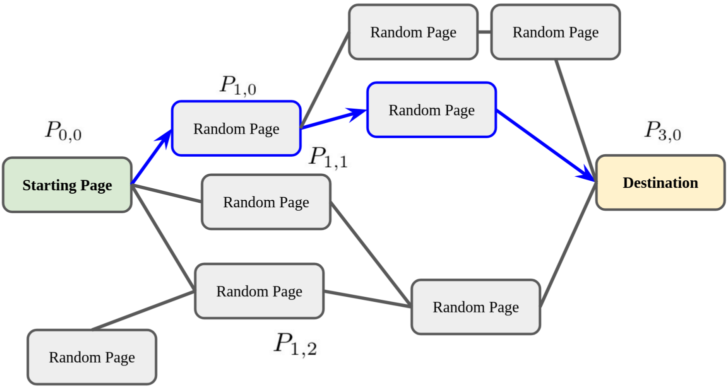


Figure: $P_{3,0}$ is adjacent to the page we've chosen, so that ends the algorithm

Formal algorithm definition

Let us denote $P_{i,j}$ as a jth page on the layer i and $\rho(P, G)$ as a distance between pages P and G.

So let us introduce the algorithm:

```
Input: Starting page $P_{0,0}$ and destination page $P_D$
Logic:
$i \leftarrow 0, j \leftarrow 0$
while $P_{i,j} \neq P_D$ do
    Form array of adjacent pages $\{P_{i+1,k}\}, k = 0, 1, \ldots$ from $P_{i,j}$
    Form array of distances $\rho_{i+1,k} \leftarrow \rho(P_{i+1,k}, P_D), k = 0, 1, \ldots$
    Find $m$ s.t. $\rho_{i+1,m} = \min\{\rho_{i+1,0}, \rho_{i+1,1}, \ldots\}$
    $i \leftarrow i + 1, j \leftarrow m$
end while
```

New Goal!

So now our goal consists in training the model to find $\rho(P, G)$ for some two pages P and G.
But that is a much easier problem since we need to train model to output a single real number instead of a full path!

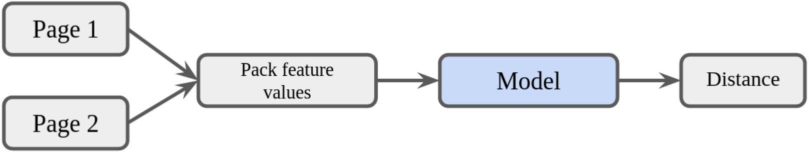


Figure: Model working principle

So what features should we give a model?

Set of Features and Outputs

1. Form array of about $40k$ "top" words and assign an integer w_k^1 $(k = 1, \dots, 40000)$ for each one.
2. Go through every word in the content of page P. If this word is contained within the list, increment the corresponding w_k^1.
3. Do the same thing for page G and set corresponding w_k^2. That way, we generated in total $80k$ inputs
4. Assign the expected minimal path $\rho_{\text{expected}}(P, G)$.
5. Train the model!

Title 1	w_1^1	w_2^1	...	w_{40000}^1	Title 2	w_1^2	w_2^2	...	w_{40000}^2	ρ_{expected}
US	1	2	...	10	Italy	2	3	...	4	3
Japan	2	5	...	23	US	3	4	...	7	1

Questions of the Day

1. How to form the set of outputs?
2. How to form the list of "top $40k$" words?

Forming a list of top words

Concept

The idea is to form the common list of words and count how many times each of them occures on our website. The numbers of appearances of each word are considered as features.

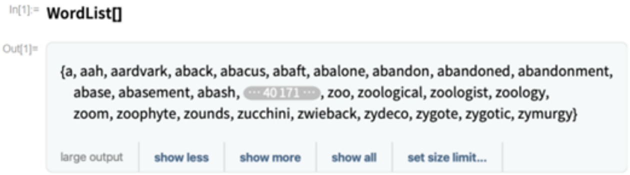
The word's selection standards:

1. The list consists of 45k words
2. The $\approx 90\%$ of the list consists of the most widespread words
3. The $\approx 0,5\%$ of the list consists of the countries' names
4. The $\approx 9,5\%$ of the list consists of the specific words, such as Reichstag, electrolysis etc. Moreover, they shouldn't be repeated with other words
5. There shouldn't be any senseless words as pronouns, articles, modal verbs etc.

Forming a list of top words

Let's consider the steps of word selection more exactly:

- To get the most common English words we used Wolfram command WordList[]



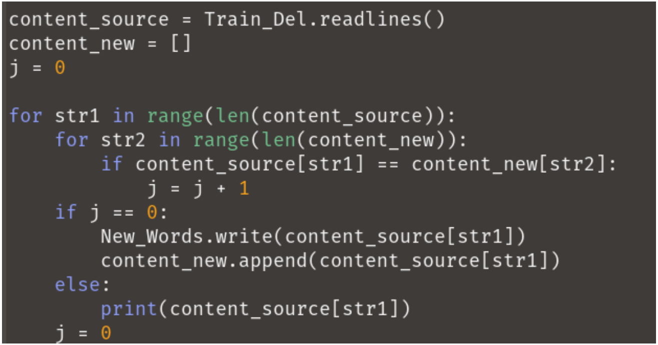
```
In[1]:= WordList[]

Out[1]= {a, aah, aardvark, aback, abacus, abaft, abalone, abandon, abandoned, abandonment,
          abase, abasement, abash, ⋯ 40171 ⋯, zoo, zoological, zoologist, zoology,
          zoom, zoophyte, zounds, zucchini, zwieback, zydeco, zygote, zygotic, zymurgy}
```

- To get specific words we used a website that gives out the related words to any topic: https://relatedwords.org/relatedto/engineer

Forming a list of top words

- To eliminate the same words we created a program which creates new file without duplications.



```python
content_source = Train_Del.readlines()
content_new = []
j = 0

for str1 in range(len(content_source)):
    for str2 in range(len(content_new)):
        if content_source[str1] == content_new[str2]:
            j = j + 1
    if j == 0:
        New_Words.write(content_source[str1])
        content_new.append(content_source[str1])
    else:
        print(content_source[str1])
    j = 0
```

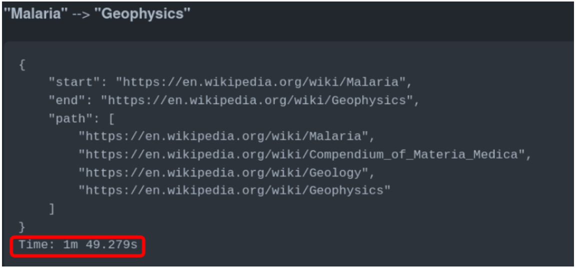
We also used this algorithm to delete senseless words

Previous algorithms

There are a couple of already implemented path finders for wikipedia pages:

- https://github.com/phrmsilva/Wikiracer
- https://github.com/stong1108/WikiRacer

But the problem with them is... Time...



"Malaria" --> "Geophysics"

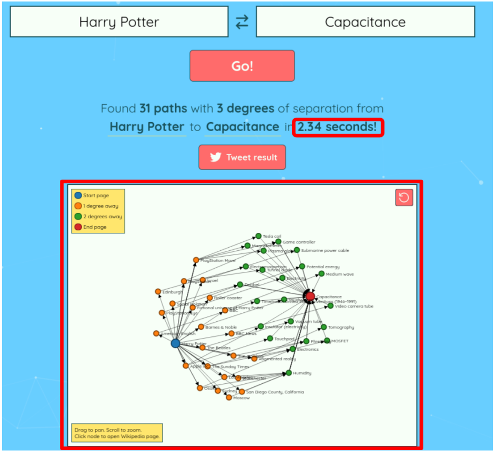
```
{
    "start": "https://en.wikipedia.org/wiki/Malaria",
    "end": "https://en.wikipedia.org/wiki/Geophysics",
    "path": [
        "https://en.wikipedia.org/wiki/Malaria",
        "https://en.wikipedia.org/wiki/Compendium_of_Materia_Medica",
        "https://en.wikipedia.org/wiki/Geology",
        "https://en.wikipedia.org/wiki/Geophysics"
    ]
}
Time: 1m 49.279s
```

Figure: Time needed for previously implemented algorithm (https://github.com/stong1108/WikiRacer) to find a path.

Problem with time

For $40k$ model features (or ultimately $20k$, spoiler) we need at least $100 - 200k$ samples. If each sample takes averaged 1 minute to load then we need to wait at least 69 days... Something definitely has to be done

Solution



Harry Potter ⇄ Capacitance

Go!

Found **31 paths** with **3 degrees** of separation from **Harry Potter** to **Capacitance** in **2.34 seconds!**

Tweet result

● Start page
● 1 degree away
● 2 degrees away
● End page

Drag to pan. Scroll to zoom.
Click node to open Wikipedia page.

Figure: Six Degrees of Wikipedia website

Solution

Why this website is useful?
Consider the picture on the left where we typed in *Apple* and *Spanish Civil War*.
We can definitely add the following info to the dataset:

Apple	Spanish Civil War	2

But using this website we may also add:

Apple	Italy	1
Italy	Spanish Civil War	1

Thus using this website, we may generate **A LOT** of data.

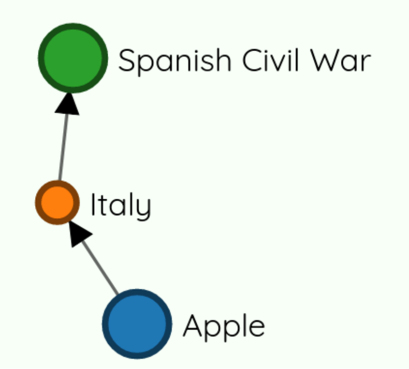


Figure: Example path graph

Data retrieval demonstration

Distances distribution

From the *Six Degrees of Wikipedia* Github page we obtained the following distances distribution:

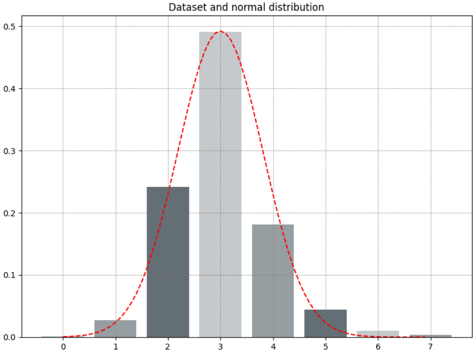


Figure: Distances distribution from the *Six Degrees of Wikipedia* statistics

Using this data, we can find the probability $P(\rho)$ of getting a distance ρ when typing in 2 random wikipages.

Thus, at the stage of adding some pair with a known distance ρ, we can add this pair with probability $p = P(\rho)$, ultimately obtaining the distribution similar to one depicted before.

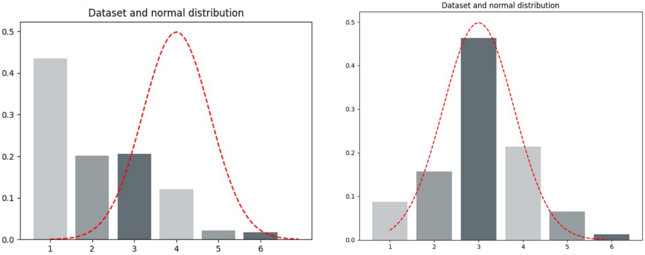


Figure: Our retrieved distribution before and after applying distribution filter

Connection between pages

We need to somehow characterize the connection between two given pages. One way how to do that is to merge two cells that correspond to w_k^1 and w_k^2, that is, find some function:

$$\varphi : \mathbb{Z}_+^2 \to \mathbb{R}$$

One way to do that is use the following rule:

1. If $w^1 = 0$ and $w^2 = 0$, $\varphi(w^1, w^2) = 0$
2. If $w^1 = 0$ and $w^2 > 0$ (or vice versa), $\varphi(w^1, w^2) = -1$
3. If $w^1, w^2 > 0$, $\varphi(w^1, w^2) = 1$

Building $\varphi(w^1, w^2)$

Notice that

$$\varphi(w^1, w^2) = \alpha(w^1)\alpha(w^2) - |\alpha(w^1) - \alpha(w^2)|,\ \alpha(w) = \frac{1 - e^{-w}}{1 + e^{-w}}$$

satisfies table of values shown before.
Another way to do that is using the function:

$$\varphi(w^1, w^2) = |\widetilde{w}^1 - \widetilde{w}^2|$$

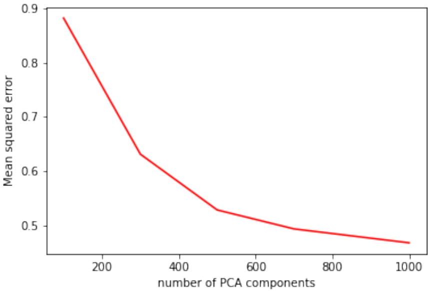
Where $\widetilde{w}^i$ is a "normalized" value of w^i (that is, we map w^i to the value $\widetilde{w}^i$ from 0 to 1).
The second option gave a better performance so we decided to stick to this formula.

Training model

During the training process, the *XGB* model gave the best performance and worked faster than another models.

To form the set of hyper-parameters, we used the *RandomizedSearchCV*.

Dimensionality reduction did not impact positively on the efficiency.



Dependency on the number of samples

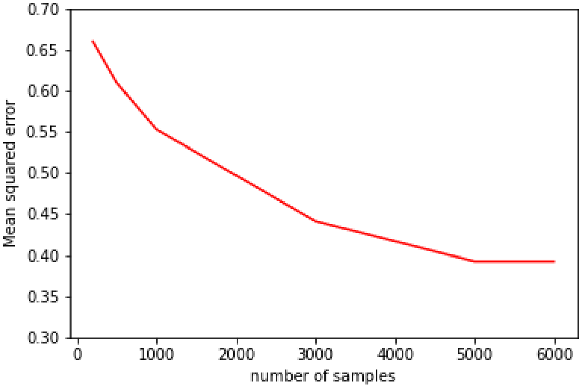


Figure: As depicted on the figure, the mean squared error is almost not affected after around 5000 samples.

Dependency on the number of features

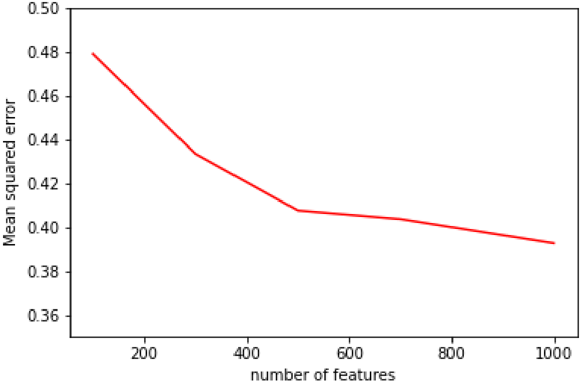
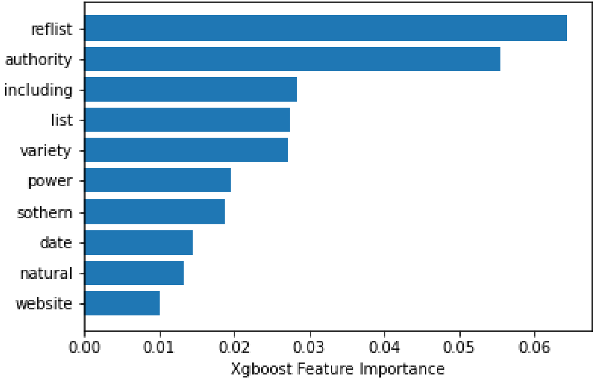


Figure: As depicted on the figure, the mean squared error is almost not affected after around 1000 features.

Most important features



Thank you for your attention!