

The WikiPlugin: A new lens for viewing the world's knowledge

CMPT733: Final Report

Matt Canute, Young-Min Kim, Donggu Lee, Suraj Swaroop, Adriena Wong

Motivation and Background

Wikipedia is the largest online encyclopedia with over 6 million English articles, increasing by over 17,000 articles every month. However, some article topics can have a wide range in difficulty, and some articles can often be overwhelming, with a seemingly endless chain of sub-topics to understand beforehand. Wouldn't it be nice to know roughly how much time you'd need in order to understand an article's main points and concepts? It would also be useful to know the minimum number of related articles you'd need to read.

Some good efforts to improve the legibility of difficult articles with overly technical terms or field-specific jargon have been made, such as the Simple English Wikipedia initiative. At the time of this writing, however, there are about [159k](#) simple-version articles out of the total roughly [6,000k](#) articles. At the same time, the [Wikimedia Foundation hosts database dumps](#) of nearly everything supporting their online content and infrastructure.

For all of this, we envisioned a tool that could highlight the important concepts of an article. We imagined that it could tell us the expected amount of time that would be necessary to read the whole page. We also imagined having a navigation view that could find similar articles not already linked within the same page, but would nonetheless contain very important contextual information that the article might expect you to already understand. We also imagined a bounty system, similar to the Stack Overflow model, that could incentivise the Wikipedia community to determine which articles to simplify next by somehow creating a priority metric based on both the difficulty and the popularity. Then we built a tool and an entire supporting infrastructure to approximately accomplish all of these ideas.

However, first we knew that in order to accomplish this, we'd need to have a system that could automatically tag how difficult an article might be, at least to a reasonable extent. A model for predicting difficulty could be useful for tools in other domains, such as in smart tutoring systems that could determine how much time might be needed to spend on some topics versus others.

Although everyone has a varying level of expertise in different domains, our intuition was that it's not hard to imagine that if you were to poll everyone on the time that it would take them to understand the concept of one article versus another, the collection of ranked lists would probably have a significant agreement with each other. We've all stumbled on articles before that could be simply understood in less than ten seconds. For example, famous person X was born in 1932, and they wrote a book about Y, and then died in 1985; that's simple and easy to understand. Compare this to one of those articles within a subject such as advanced quantum mechanics, where the majority of us considers it to be a particularly difficult subject to understand, because it relies on a massive foundation of other knowledge that we'd be expected to know, and it's filled with rare jargon

everywhere. What if the graphical information of Wikipedia, such as the typical length of a possible path to reach an article, was a useful predictor of how much foundational knowledge was necessary? What if the term frequency and inverse document frequency could mechanically discover rare jargon words as another predictive feature? Could we use topic modelling to figure out when an article is simply describing a famous person's life?

Problem Statement

Our goal with this project was to address the following two questions:

1. Are there certain features or patterns of an article that could allow an analytical model to reasonably learn how to predict whether or not the article's concept would be difficult to understand?
2. If so, how could we best use that tool to provide an overlay of helpful information, and to guide the reader's experience throughout the article?

While there has previously been [similar work](#) done on predicting the reading difficulty of an article by using a language modeling approach, we were unable to find a study that leveraged both the massive text sources *and* the graphical representation of Wikipedia in order to do so.

This problem was especially challenging due to the following issues:

- The datasets we were interested in were both very large and unstructured. The underlying graphical nature of the monthly 33 million rows of click volume data would suggest that it should be stored into a graphical database format, such as neo4j. However, the accompanying text data for each article ought to be stored in a document database as some articles could range from [21K words](#) to [1.6K words](#), and that's not even the max/min range. We had to put quite a bit of thought into deciding what the best data architecture would be in order to begin pre-processing it for analytical purposes.
- Even if we were to store the click volume data in a neo4j database, and the text data in a document database, how would we efficiently join those two different sources of information together, given that they were in very different data storage representations?

Data Science Pipeline

The two large datasets mentioned above were behind the entire pipeline of building the machine learning model to classify article difficulty, and this same pipeline was turned into an officially scheduled ETL pipeline for the model to score new articles on a monthly basis. For that reason, describing the ETL pipeline would be analogous to describing the final iteration of the feature extraction for our model-building purposes. First, we'll describe the datasets in a bit more detail.

Article Text

This is the [monthly snapshot of all of the text](#) found on Wikipedia. It's represented in a compressed XML format. You'll notice that it is currently split into 27 different datasets of roughly the same file size, which makes it a great candidate for a map-reduce flavour of ETL, given the proper server

resources.

Clickstream

This is the [monthly snapshot of the Wikipedia clickstream dataset](#), which contains information about the counts of Wikipedia hyperlink clicks extracted from its request logs. It contains the article title and the page from which someone clicked from, whether it's from another Wikipedia article or from external sources, such as from a Google search. This dataset contains around 33 million rows each month.

We have a Docker/Jenkins script that runs an automated job every month to perform the following tasks:

1. Downloading the latest datasets

The English articles text datasets are split into multiple streams from the Wikimedia dumps websites which helps with parallelization, especially useful for data cleaning and preprocessing. The clickstream dataset is downloaded as one large file every month.

2. Cleaning and preprocessing

Cleaning/preprocessing the English-language Wikipedia dumps involves the following tasks:

- Keeping only redirect destination Wikipedia links / duplicate removal
- Removing HTML elements from the text
- Extracting all hyperlinks and keeping only Wikipedia-to-Wikipedia (internal) links
- Changing all text to lowercase
- Removing stopwords downloaded from the NLTK package
- Keeping only alpha characters (removing punctuation and numerical values)
- Mapping each word to a word vector based on a trained Word2Vec model
- Calculating the average word vector of the article

Cleaning the clickstream dataset involves the following:

- Keeping only article names with english alphabet characters, numbers, -, and _
- Dropping any null values
- Counting the number of internal versus external requests

3. Feature extraction for the difficulty model

Text Metric Calculations - The Flesch-Kincaid Reading Ease score is a text heuristic to approximate how difficult an English sentence might be to understand. We used the [textstat](#) package to generate this score for every article.

Latent Dirichlet Allocation (LDA) Topic Modelling - This generates a distribution of the five most common Wikipedia topics for each article. This leveraged the [gensim](#) Python library to build a dictionary, a word corpus, and a cached LDA model for the articles. It then gets reloaded and

calculates the topic distribution for each article.

Graphical Metrics Calculations - For each article, we calculated the degree and the eigenvector centrality using the [networkx](#) package.

4. Scoring the data

We then used the previously discussed features to train a difficulty model and cached it in order to determine a difficulty score for the rest of the Wikipedia articles. This model was trained, tested, and cached on our server using the previously discussed features on a manually tagged set of 300 articles to assign a difficulty score based on our own subjective opinions. More details about this process can be found in the Methodology section.

5. Uploading summarized datasets to the Cloud SQL database

Since we've effectively reduced the size of the datasets down by more than a factor of 10 while still keeping aspects of the graphical importance and semantic information for each article, we can load the latest month of the clickstream and article text representation in the form of vectors onto a Google Cloud SQL database with careful indexing on the article name so that the performance is still reasonable.

6. Generating separate files for the Chrome extension

A SQLite file was created for the Chrome extension. This file contained information about the clickstream for each hyperlink from the previous month as well as the estimated read time (minutes) per article.

Methodology

As mentioned previously, we randomly sampled 300 articles from Wikipedia to manually label its difficulty in order to train the difficulty model through supervised learning. We did this while following these general guidelines:

1. Try to read the entire article for up to 2 minutes
2. Assign the article as belonging to one of these three groups:
 - (a) The article is completely easy to understand, and I could write a summary of the overall concept in a few sentences or less.
 - (b) I wasn't able to finish the article within 2 minutes, but I think I could understand it. However, I'd need to click some related links and/or spend more time to understand it.
 - (c) This article is complicated, and probably even after spending another 20-30 minutes on it, and/or related links, I still wouldn't be able to confidently write a summary about it.

We then grouped (b) and (c) together as 'difficult' whereas the articles labelled as (a) were considered

to be 'easy'. This dataset was coincidentally evenly split between easy and difficult articles. We then used these scores along with the other features to train a logistic regression model.

For topic modelling to feed into the difficulty model, we experimented with a range of different topics, and found that since we wanted to keep the number of features low relative to the low volume of the training set, we found that five was the lowest number we could use while maintaining reasonable interpretability. These topics consisted of the following: social studies related articles (history, geography, politics, etc), hard sciences (math, physics, chemistry), technical (internet protocols, hardware), dates (indexing time periods, meta), and famous people (authors, singers, actors).

We had to get creative with some other features as well. For example, to calculate the eigenvector centrality measure for all articles in a reasonable length time, we split up this workload into a map-reduce sort of framework. This meant approximately building a few thousand local graphical networks that different nodes could work on, from randomly sampled nodes and edges (articles, and article click-throughs) and then joining up the results afterwards in one large table. This means that the feature was an approximation, however it was applied consistently for each article.

Evaluation

In order to evaluate our difficulty model, we used 80 manually tagged articles separate from the training set to compare with the predictions generated from the model. Pictured below is the confusion matrix of that test set:

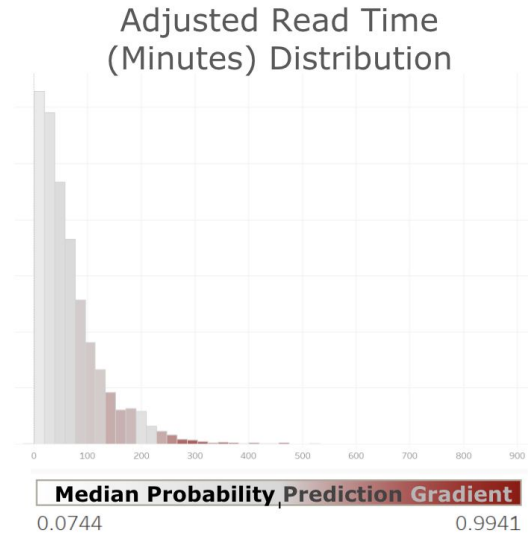
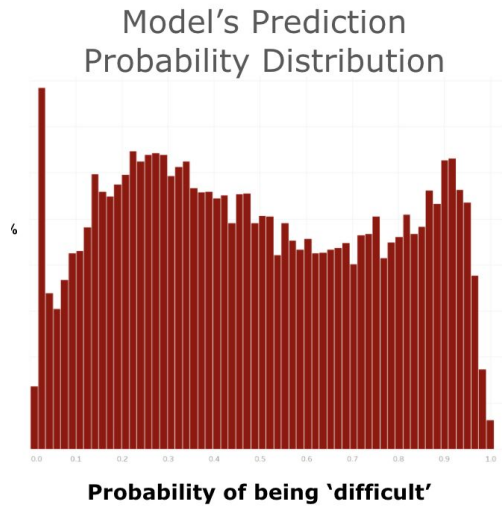
Overall **Confusion Matrix** on Test Set

	Predicted		
Actual		Not Difficult	Difficult
	Not Difficult	15	7
	Difficult	19	39

Overall Accuracy at .5 Threshold: **67.5%**

Precision at top 20%: 26/28 **93%**

After inspecting the coefficients of the features with the prediction, the length of the text was a strong predictor, as well as a collection of columns representing certain dimensions in the word vector centroid representation of the article, which was less interpretable, but the topic distribution for articles in the hard sciences was also as strong predictor. Scoring all the articles overall, it appeared that the distribution of difficulty was bi-modal, as pictured below on the left. Also pictured below to the right is the distribution of the adjusted read time in minutes with the median difficulty probability score shaded according to a gradient between .07 and .99, so we can see that there are certain cases where our tool's estimated read time was more strongly adjusted than others due to the additional probability score calculation.



Potential model improvements:

The easiest option for improving our model performance would be to increase our dataset by manually labelling more articles, especially on articles where the current model's prediction probability is close to the .5 boundary.

Another possibility would be to experiment with a larger difficulty range in our article tagging exercise. Currently, our three initial labels can be thought of as a 3-point difficulty scale, but it might be beneficial to experiment with a range of 5 or 10.

Our accuracy could also be improved if each article were to be tagged by multiple people. Then we could take the majority label as well as the label variance into account so that we could fit the data onto a weighted logistic regression model instead.

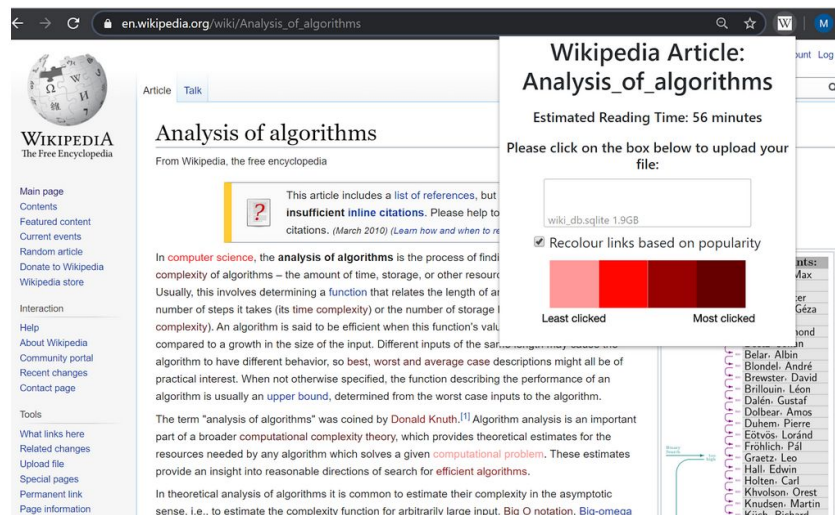
We could also experiment with other model architectures later on. The reason that we stuck with a simple logistic regression for this project was so that we could have something simple and interpretable to work with in the overall pipeline of the project, and it would be easy to enhance later on.

Data Products

Chrome extension:

Wikipedia users can point to the necessary SQLite file within their extension, and then they're able to use this tool within Chrome. This browser extension helps readers to identify the most relevant articles on a page by recolouring all of the hyperlinks based on the historical click volume, effectively crowdsourcing the latest month's clicks to generate a heatmap of the typical reader path from this article. It also displays each article's estimated read-time. The reading time estimate is calculated by the standard method of taking the number of words in an article, and dividing that by 225 (since the average person reads 200 - 250 words per minute), but then multiplying this by a factor of 1 + the

probability that the article would be tagged as 'difficult'. We're open to improvements of this simple calculation approach to reading time estimates in later versions of the tool.



The Chrome extension with the article links recoloured

A hyperlink is also provided on the plugin popup that will direct users to our WikiPlugin Analysis webpage, which contains three views: the Drill-Down page, the Summary page, and the Simple Priority page. These are described in more detail below.

The Drill-Down page:

Users can enter a specific article name into the textbox, such as "Supervised Learning". This creates two tables to display the Graphical Similarity and the Semantic Similarity of related articles. The Graphical Similarity table shows the top articles based on the most popular paths to reach this article, which is basically like what the heatmap is visualizing, but in the opposite direction. I.e., the plugin's heatmap visual colours the links based on the clicks to those links from the page you're on, whereas this table shows the top originating pages that were clicked on in order to reach the article of interest. The Semantic Similarity on the other hand shows the top articles that are the most semantically close to the article of interest based on their article text, using their word vector centroid distances. Contrasting the semantically similar articles that don't already belong to the graphical similarity table should ideally allow you to find related analogous concepts, even though they may belong to a different field of study. We also display each article's estimated read time as measured in minutes.

Graphical Similarity

No.	Title	Link
1	Machine_learning	https://en.wikipedia.org/wiki/Machine_learning
2	Reinforcement_learning	https://en.wikipedia.org/wiki/Reinforcement_learning
3	Unsupervised_learning	https://en.wikipedia.org/wiki/Unsupervised_learning
4	Deep_learning	https://en.wikipedia.org/wiki/Deep_learning
5	Support-vector_machine	https://en.wikipedia.org/wiki/Support-vector_machine

Semantic Similarity

No.	Title	Link	Distance	R. Time
1	Numerical_analysis	https://en.wikipedia.org/wiki/Numerical_analysis	0.9637356142825492	150.424
2	Functional_decomposition	https://en.wikipedia.org/wiki/Functional_decomposition	0.97676946910157	116.828
3	Signal_separation	https://en.wikipedia.org/wiki/Signal_separation	1.0681614649075286	28.1562
4	Analysis_of_algorithms	https://en.wikipedia.org/wiki/Analysis_of_algorithms	1.0771085641041296	55.3024
5	Search_algorithm	https://en.wikipedia.org/wiki/Search_algorithm	1.0845344559556997	56.3308

The tables on the Drill-Down page

The Summary page:

Users can enter the desired month and year, such as '202001,' and this creates a table showing the aggregated five topics based on our LDA topic modelling originally used for our difficulty model. This is generally a placeholder where we would put additional analysis later on to show interesting surges of interest over time, similar to google's [trends](#) service.

Summary for 202001

No.	Title	Proportion
1	Social Studies	29.076725417927342
2	Hard Science	19.1825218459174
3	Dates	6.04651536682204
4	Technical	15.187901049005287
5	Celebrities	30.506336320327936

The table on the Summary page

The Simplification Priority Queue page:

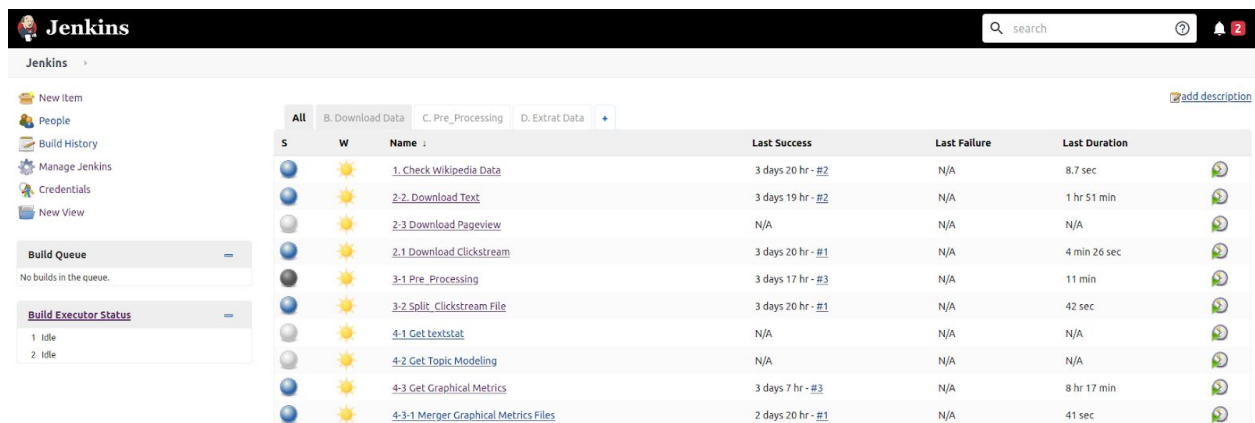
This simplification priority queue could be very useful for the Wikimedia group to determine which articles are most in need of having a simplified version. For all the pages that don't already have a simplified view, we have a rough estimate of the difficulty score on millions of articles, and have a proxy for demand through the historical click volume. This could be used by the community of volunteers to prioritize which articles should be simplified next, and they could perhaps have a bounty system from the top items in the queue, similar to the Stack Overflow [incentives model](#).

Simplification Priority

No.	Title	Link	Difficulty	Read Time (min)
1	Arabic	https://en.wikipedia.org/wiki/Arabic	0.999785	405.075
2	List_of_Latin_words_with_English_derivatives	https://en.wikipedia.org/wiki/List_of_Latin_words_with_English_derivatives	0.994149	1160.93
3	List_of_compositions_by_Johann_Sebastian_Bach	https://en.wikipedia.org/wiki/List_of_compositions_by_Johann_Sebastian_Bach	0.992929	753.098
4	Complexity_theory	https://en.wikipedia.org/wiki/Complexity_theory	0.991252	4.64616
5	Cognitivism	https://en.wikipedia.org/wiki/Cognitivism	0.989095	5.33644

The table on the Simple Priority page

Underneath all this is a Jenkins pipeline to update the cloud database on a monthly basis. Instructions for setting this up within a docker container on any server can be found within the README.md file of the repository.



S	W	Name	Last Success	Last Failure	Last Duration
1	Success	1. Check Wikipedia Data	3 days 20 hr - #2	N/A	8.7 sec
2	Success	2. Download Text	3 days 19 hr - #2	N/A	1 hr 51 min
3	Success	2-3 Download Pageview	N/A	N/A	N/A
4	Success	2.1 Download Clickstream	3 days 20 hr - #1	N/A	4 min 26 sec
5	Success	3-1 Pre Processing	3 days 17 hr - #3	N/A	11 min
6	Success	3-2 Split Clickstream File	3 days 20 hr - #1	N/A	42 sec
7	Success	4-1 Get textstat	N/A	N/A	N/A
8	Success	4-2 Get Topic Modeling	N/A	N/A	N/A
9	Success	4-3 Get Graphical Metrics	3 days 7 hr - #3	N/A	8 hr 17 min
10	Success	4-3-1 Merger Graphical Metrics Files	2 days 20 hr - #1	N/A	41 sec

Our Jenkins pipeline

Lessons Learned

This project definitely gave us the opportunity to learn and try things that we haven't done before. For instance, it was our first time making a Google Chrome extension so there was a small learning curve that came with building it from scratch, especially given that we wanted to connect our large SQL database to it. We ended up using code from <https://sqlreader.freebusinessapps.net/> to connect the extension to the database and it was a good learning experience to try to fit that into our project's context.

One of the concepts we had minimal experience with prior to this project was using Jenkins with Docker. It wasn't simple to put Jenkins into a Docker container but we felt that having this would make it easier to manage and test all the necessary tasks in our pipeline in one simple view, such as downloading the data, cleaning the data, and other ETL tasks mentioned above. Although we only set

it up on a local machine, it can be scaled and expanded if we had a larger budget and perhaps more demanding tasks.

The greatest takeaway probably would have to be the experience of emotionally realizing the necessity of having the appropriate data storage design to derive useful analytical insights, rather than just intellectually knowing about such techniques. Without really struggling with a large dataset, learning big data concepts can often feel like learning how to kill mosquitoes with bazookas; in this case however, we had to be very thoughtful in the design of our data structures, because otherwise the tool just wouldn't work. The drill-down page for example would have had terrible performance with loading times that could last until the end of the universe if we weren't careful.

In summary, each of us gained both more practical experience, and a better understanding of the following tools:

1. Docker
2. Jenkins
3. Flask, sklearn, gensim, numpy, pandas, NLTK, TextStat, NetworkX
4. Google Chrome Extensions, HTML, Javascript
5. MySQL (Google Cloud), SQLite

Summary

In the end, we were mostly able to build the tool that we originally imagined. Our infrastructure design for the ETL pipeline to update our back-end system on a monthly basis was very carefully considered. Our database and our web application are up and running, supporting the home page of our plugin, allowing a user to highlight the important concepts of an article and to see the expected time required to read the whole page. They can find similar articles to the one that they're trying to learn about, and can use the two views to potentially find analogous concepts in other subjects. We're excited to continue this project further after the course and are interested to see where it takes us.