**About**

This website aims to provide an overview of some simple Natural Language Processing (NLP) techniques to analyse text. Hopefully, it’ll give you a flavour of NLP and maybe even spark an interest!

**What is NLP?**

NLP is a branch of artificial intelligence that allows computers to interpret and extract insights from human language. This draws from methods in linguistics, computer science and mathematics. The field of NLP has grown rapidly in recent year with an ever-increasing volume of textual data and with improvements in NLP techniques. Popular applications include monitoring brand sentiment, summarising news articles and analysing social media posts.

**Website format**

Each web page will explain one NLP technique. The pages are split into two sections:

1. **Demo section:** At the top you can interact with the technique live. Hopefully this will help to make the technique less abstract when you read the explanation.
2. **Explanation section:** The technique will be explained underneath the demo section. I’ve tried to keep the explanations simple while providing links to advanced material for those wishing to explore further. Each explanation section will outline 1) what the technique is, 2) approaches to implementing the technique, 3) how the implemented approach works and 4) the limitations of the approach.

**Rationale for building**

I’m a Data Scientist with a passion for NLP. I primarily use Python as my default scripting language, but recently expanded my scripting stack to include HTML, CSS and JavaScript – the building blocks for web development.

I built this website to solidify my HTML, CSS and JavaScript knowledge while hopefully explaining some simple NLP techniques to those not familiar with the field.

If you want to get in touch my LinkedIn is probably the best shout ☺ It’s linked below (no pun intended).

Paresh Sharma [06/01/21]

[PHOTO]

**Text Summariser**

**What is automatic text summarisation?**

Automatic text summarisation is a natural language processing technique to produce summaries from long text. Real world applications include summarising news articles/reviews/search engine results and checking the key points in long text.

**Methods for automatic text summarisation**

The overwhelming volume of information in society has led to the desire of many researchers to develop approaches to automatically summarise text into meaningful information.

There are two main approaches – extractive and abstractive summarisation.

**Extractive** summarisation extracts the most important key-phrases/sentences from text and combines them to produce a summary. **Abstractive** summarisation paraphrases text to produce a summary.

To illustrate the difference:

**Original text** (from Wikipedia) – “The Japanese macaque (*Macaca fuscata*), also known as the snow monkey, is a terrestrial Old World monkey species that is native to Japan. They get their name "snow monkey" because some live in areas where snow covers the ground for months each year – no other non-human primate is more northern-living, nor lives in a colder climate. Individuals have brownish grey fur, pinkish-red faces, and short tails. Two subspecies are known.”

**Extractive summary** - The Japanese macaque (*Macaca fuscata*), also known as the snow monkey, is a terrestrial Old World monkey species that is native to Japan. Individuals have brownish grey fur, pinkish-red faces, and short tails.

**Abstractive summary** - The Japanese macaque (*Macaca fuscata*) is a monkey species native to Japan which has brownish grey fur, pinkish-red faces, and short tails.

As you can see, both methods work well, but abstractive summarisation is closer to mimicking human summarisation. This is because it uses more complex models which can better capture the nuances of natural language. That said, there are several downsides of models that use abstractive summarisation 1) long training time, 2) large storage requirement and 3) long text summarisation time.

**Extractive and abstractive models**

There are several different models which can be used for extractive and abstractive summarisation. Popular extractive models include: *Word Frequency, Non-Negative Matrix Factorisation, Latent Semantic Indexing* and *TextRank*. Popular abstractive models include: *BERT, T5, XLNet, RoBERTa, ALBERT* and *GPT-3*. ADD LINK

For my text summariser, I originally planned to implement an abstractive model. Unfortunately, there were a few of constraints which made this impractical and infeasible. I have eluded to some of the constraints with abstractive models in the previous section (for more detail see section below: Practical implementation problems with abstractive models**)**.

Resultantly, I used the extractive model TextRank.

**How does TextRank work?**

TextRank ranks sentences in text by importance and outputs the most important sentences as the summary. Importance is determined by a sentence’s similarity to other sentences. I will explain the algorithm step by step below and include a simplified example.

Given some text:

1. Split the text into individual sentences.
2. Find the word embedding (vector representation) for each sentence.

Word embedding describes a technique that maps words to numbers in an N-dimensional vector space.

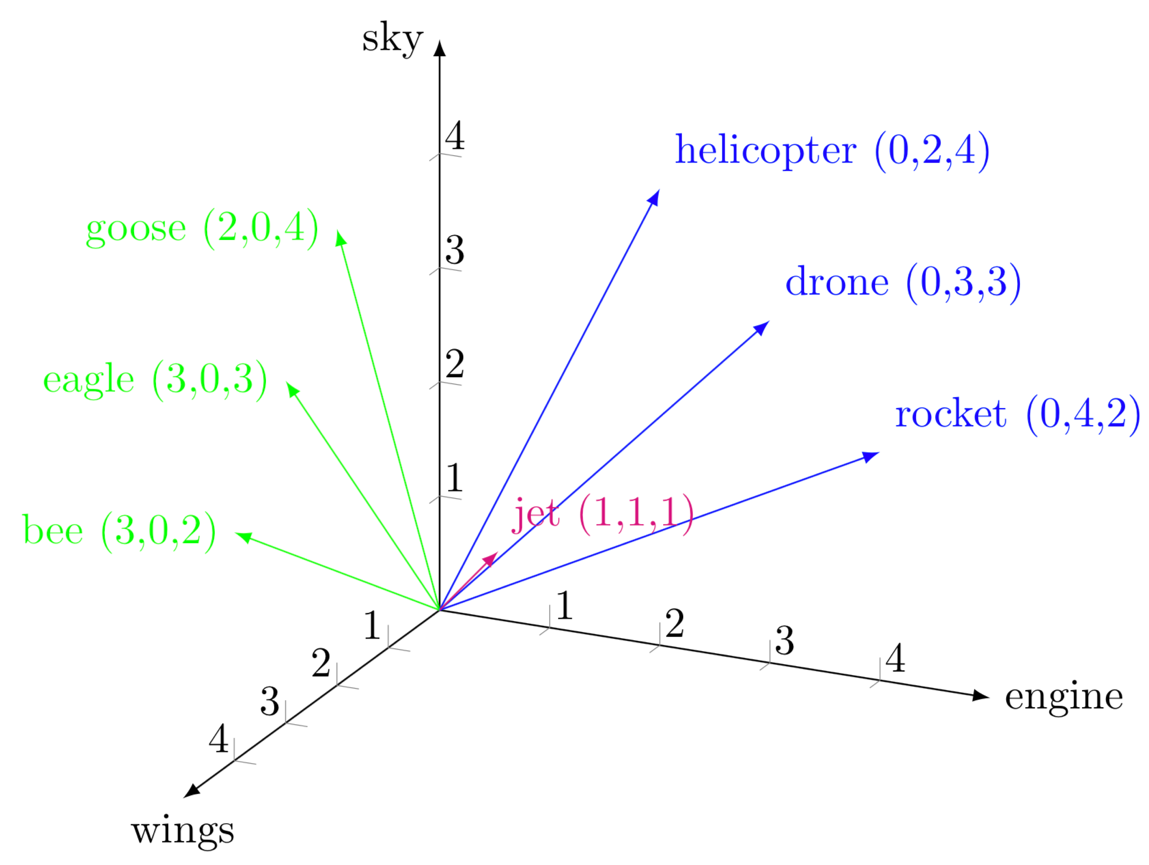
Figure 1 represents a 3-demensional vector space. Each word has a 3-dimensional vector representation e.g. “helicopter” is represented by (0, 2, 4). The distance between vectors represents how similar the words are. A smaller distance indicates words with similar meaning. There are a variety of methods to calculate this similarity score between two vectors - one of the most popular methods is the cosine distance.

In a similar fashion, sentences can also be represented in an N-dimensional vector space. Sentences with similar meanings are mapped to a similar vector space and similarity scores can be computed.

The example provided is a simplification. In practice, word embeddings map most of the words in the English dictionary to hundreds of dimensions. I will not go into how word embeddings are calculated, but if you are interested please click here. ADD LINK

Figure 1. Word embeddings in a 3-dimensional vector space. Reference: Guillaume Desagulier, "Word embeddings: the (very) basics," in *Around the word*, 25/04/2018, <https://corpling.hypotheses.org/495>

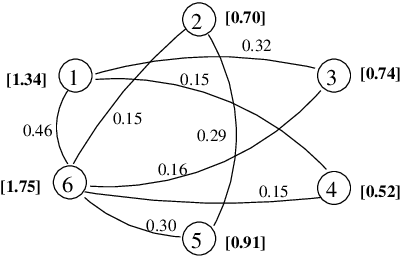
<https://www.google.com/url?sa=i&url=https%3A%2F%2Fcorpling.hypotheses.org%2F495&psig=AOvVaw0NXxd2gpsvrr8BkSw_--bo&ust=1609330286010000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCJiiidSU8-0CFQAAAAAdAAAAABAD>



Guillaume Desagulier, "Word embeddings: the (very) basics," in *Around the word*, 25/04/2018, <https://corpling.hypotheses.org/495>

1. Calculate the vector similarity scores between sentences and store in a *similarity matrix* (ADD A LINK). This records the similarity between all of the sentence.
2. The similarity matrix is converted into a network graph (see Figure 2), with sentences as vertices (circles) and similarity scores as edges (connections between circles).
3. The PageRank algorithm is used to calculate the sentence rank. Sentences will be ranked higher if they are more similar to other sentences. I won’t go into the specifies of the PageRank algorithm, but if you’re interested click here (ADD LINK).
4. The top X ranked sentences are output as the text summary.

Figure 2. Network graph of sentence similarities. Scores reflecting sentence importance are shown in brackets next to each sentence.



<https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.researchgate.net%2Ffigure%2FGraph-of-sentence-similarities-built-on-a-sample-text-Scores-reflecting-sentence_fig1_228340005&psig=AOvVaw2zRv8qZ4GyS7eWcgHZokT4&ust=1609337098945000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCKDCvoOu8-0CFQAAAAAdAAAAABAm>

<a href="https://www.researchgate.net/figure/Graph-of-sentence-similarities-built-on-a-sample-text-Scores-reflecting-sentence\_fig1\_228340005"><img src="https://www.researchgate.net/profile/Paul\_Tarau/publication/228340005/figure/fig1/AS:650513741774852@1532105970144/Graph-of-sentence-similarities-built-on-a-sample-text-Scores-reflecting-sentence.png" alt="Graph of sentence similarities built on a sample text. Scores reflecting sentence importance are shown in brackets next to each sentence."/></a>

A language independent algorithm for single and multiple document summarization - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Graph-of-sentence-similarities-built-on-a-sample-text-Scores-reflecting-sentence\_fig1\_228340005 [accessed 29 Dec, 2020]

**Note:** The algorithm I am using is actually a variation of the TextRank algorithm. The steps are the same, but the similarity score is calculated via the BM25 algorithm rather than cosine distance (as it improves the text summarisation).

**Limitations of TextRank**

* It outputs the most important sentences; this is not how humans summarise information.
* It cannot summarise small amounts of text well.

**Practical implementation problems with abstractive models (Optional)**

I originally planned to implement the BERT abstractive model instead of the TextRank extractive model.

BERT is a state-of-the-art machine learning model for text summarisation. The model works best when trained on a huge text-based dataset to learn the contextual relationships between words.

I really wanted to showcase BERT in this web application, but unfortunately, I ran into a few problems synonymous with abstractive models:

**1) Long training time**

Training a model of this nature generally requires multiple high-performance GPUs in the cloud. This can be very expensive. Furthermore, this is an energy intensive process which leads to environmental costs. To limit these problems machine learning researchers have open-sourced pre-trained models such as BERT so others do not need to burden the brunt of the economic or environmental costs. Therefore, I was able to overcome this issue by using a pre-trained BERT model.

**2) Large storage requirement**

The pre-trained BERT model was trained on the 800GB+ Colossal Clean Crawled Corpus. This dataset consists of raw web page data, extracted metadata and text extractions from the web. BERT uses this to learn the contextual relationships between words in text and stores that information as “pre-trained weights” also sometimes referred to as “parameters”. We can download these parameters to use BERT. Depending on which version of BERT you are using the number of parameters can range from 30 million (BERT small) to 11 billion (BERT large). The storage space for small and large BERT are 231.1MB and 42.1GB respectively.

I am hosting my web application via a service called Heroku. The lite version of Heroku offers 512MB RAM storage space. As you may have guessed BERT large is way too big to store on my lite account! Of course, I could have upgraded my lite Heroku instance to increase the RAM, but that could get expensive.

That said, I managed to overcome this issue by using BERT small.

**3) Long text summarisation time**

Heroku and other similar platforms tend to have a server response timeout limit to maintain good customer service. This is usually 30 seconds, meaning that the web application crashes after this limit is reached. BERT small takes about 40 seconds to 1 minute to summarise text (as it uses 30 million parameters). As you may have guessed, my server crashed. I have a couple of ideas for workarounds, but they’re quite time consuming. Given that this is a side project I opted to sub-out BERT and sub-in TextRank. TextRank (as is the case for other extractive methods) is a much smaller model to store and has no pre-trained weights, thus the text summarisation is much faster.

If you’re interested in implementing BERT or just want to have a play feel free to reach out and I can point you in the right direction!

A picture containing text

Description automatically generated

**References:**

BERT paper: <https://arxiv.org/abs/1810.04805>

XLNet: <https://arxiv.org/abs/1906.08237>

RoBERTa: <https://arxiv.org/abs/1907.11692>

ALBERT: <https://arxiv.org/abs/1909.11942v1>

GPT-3: <https://arxiv.org/abs/2005.14165v2>

T5: <https://arxiv.org/pdf/1910.10683.pdf>

Word frequencies paper: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.3248&rep=rep1&type=pdf>

Non-Negative matrix factorisation: <https://www.sciencedirect.com/science/article/abs/pii/S030645730800068X>

Latent Semantic indexing: <https://dl.acm.org/doi/10.1145/383952.383955>

TextRank: https://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf

Gensim TextRank implementation: <https://arxiv.org/pdf/1602.03606.pdf>

Word embeddings: <https://arxiv.org/pdf/1310.4546.pdf>

Similarity matrix example: <https://www.researchgate.net/figure/Example-of-similarity-matrix_fig3_315628119>

PageRank example video: <https://www.youtube.com/watch?v=qtLk2x59Va8&ab_channel=RSREETech-NLP%2FAI%2FMLsimplified>

**Sentiment Analysis**

**What is sentiment analysis?**

Sentiment analysis is a natural language processing technique to determine whether text has positive (+1), negative (-1) or neutral sentiment (0). Popular real-world applications include monitoring sentiments in brands, customer feedback and emails.

Here is a simple example to illustrate positive, neutral and negative text: **(ADD COLOURS TO EACH)**

“I love chocolate”

“Chocolate is alright”

“Chocolate is awful”

**Approaches to sentiment analysis**

There are two main approaches – lexicon-based and machine learning.

**Lexicon-based** approaches aim to build a collection of words associated with a particular sentiment orientation (positive, negative and neutral). Each word is given a polarity score. This lexicon is used to score the sentiment of text. Researchers have built a variety of sentiment lexicons such as Valence Aware Dictionary and sEntiment Reasoner

(VADER) and TextBlob. **(ADD THE LINKS)**

**Machine learning** approaches build algorithmic models which can classify sentiment in text. Models are trained on a large corpus of pre-labelled sentiment orientated text. Common models include: Naives Bayes, Support Vector Machines and Long-Short Term Memory Networks. **(ADD THE LINKS)**

Machine learning approaches are more accurate as they better capture the indicators of sentiment in text. Unfortunately, they have several downsides 1) long training time, 2) large storage requirements and 3) long sentiment scoring time. This can make them difficult to use in a small web application like mine. I will not explain why as there is a similar discussion on the Text Summarisation page (ADD LINK) under the section “Practical implementation problems with abstractive models”**.**

Despite the superior performance of machine learning approaches, I employed the lexicon-based approach. Specifically, I used the TextBlob Lexicon. I chose this lexicon over other options as it performs well for formal text (vs text from social media etc) and is simple to implement.

**How do you build a sentiment lexicon and score the sentiment of text?**

**Building a sentiment lexicon**

A sentiment lexicon consists of hundreds/thousands of words associated with positive, negative and neutral sentiment. Each word has a polarity score (between -1 to 1) to indicate sentiment strength. For example, “good” may have the score 0.7 whilst “great” may have the score 0.8.

How are these words and scores obtained? There are several methods 1) computational linguistic experts manually score words, 2) machine learning algorithms assign scores to words by training on sentiment annotated text, or 3) they are estimated via word ontologies. Popular pre-defined word and score lists include: SentiWordNet, AFINN and labMT. (ADD LINKS).

**Scoring the sentiment of text**

With a lexicon built, sentiment scores can be calculated for text. This may differ depending on implementation. Since I used TextBlob I will explain how this score is calculated.

TextBlob uses simple averaging along with other special case rules to express sentiment. Let’s break it down.

1. **The base case - simple averaging**

Assume the words “jail” and “good” have polarity scores -0.1 and 0.7 respectively. If all other words have a polarity score of 0, the sentence “My jail is good” will use the average polarity i.e. [-0.1 + 0.7]/2 = 0.3. (USE JS LIBRARY)

1. **Special cases - Negation, modifiers, 1 letter words and unknown words**

If we change the sentence to “My jail is not good” the sentence turns negative. The word “not” **negates** the positive sentiment of the word “good”. TextBlob recognises this and uses a negation multiplier of -0.5 to multiply the polarity of the word “good”. The sentence’s polarity is given by [-0.1 + (-0.5)\*0.7]/2 = -0.225. (USE JS LIBRARY)

If we change the sentence to “My jail is very good” the sentence becomes more positive. The word “very” **modifies** the positive strength of the word “good”. TextBlob’s lexicon not only captures the polarity score, but also the intensity score. The intensity score determines if a word modifies the next word. TextBlob recognises “very” as a modifier word and ignores it’s polarity score (0.2) and uses it’s intensity score (1.3) to multiply the following word. The sentence’s polarity is given by [-0.1 + (1.3\*0.7)]/2 = 0.405. Interestingly, this same logic also applies for punctuation modifiers like exclamation marks (you can try this yourself above!). (USE JS LIBRARY)

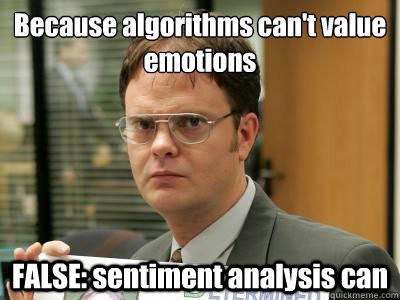
If we change the sentence to “My jail is not very good” we have a **negation combining with a modifier**. In this instance, the polarity of the word “good” is multiplied by -0.5 (for the negation) but now the inverse intensity (1/intensity) score is used. The sentence’s polarity is given by [-0.1 + (-0.5\*(1/1.3)\*0.7)]/2 = -0.185. (USE JS LIBRARY)

TextBlob **ignores 1 letter words and words not in the lexicon**.

**Limitations of sentiment with lexicons**

Sentiment lexicons use individual words to determine sentiment and not the context. Thus, they cannot capture irony, humour and sarcasm.



**Look for sentiment analysis memes – look for another**

**References:**

WKWSCI Sentiment Lexicon: <https://blogs.ntu.edu.sg/chriskhoo/2017/07/wkwsci-sentiment-lexicon-v1-1-available-for-download/>

Vader lexicon: <https://github.com/cjhutto/vaderSentiment/blob/master/vaderSentiment/vader_lexicon.txt>

NRC lexicon: <https://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm>

Textblob: <https://github.com/sloria/TextBlob/blob/eb08c120d364e908646731d60b4e4c6c1712ff63/textblob/en/en-sentiment.xml>

Naive bayes and SVM: <https://arxiv.org/pdf/cs/0205070.pdf>

LSTMs: <https://arxiv.org/pdf/1512.01100.pdf>

AFINN lexicon: <https://github.com/fnielsen/afinn/blob/master/afinn/data/AFINN-en-165.txt>

SentiWordNet lexicon: <https://github.com/aesuli/SentiWordNet/blob/master/data/SentiWordNet_3.0.0.txt>

labMT lexicon: <https://pydigger.com/pypi/labMTsimple>

SentiWordNet paper: [**https://github.com/aesuli/SentiWordNet**](https://github.com/aesuli/SentiWordNet)

**Subjectivity Analysis**

**What is subjectivity analysis?**

Subjectivity analysisis a natural language processing technique to determine the extent to which text is based on facts (objective) vs opinions (subjective). Facts are statements that can be proven right or wrong. Opinions are expressions of a person’s feelings that cannot be right or wrong.

Real-world applications include improving answer quality in online forums and analysing social media posts/politician’s speeches.

**Approaches to subjectivity analysis**

There are two main approaches – lexicon-based and machine learning.

**Lexicon-based** approaches aim to build a collection of words associated with subjectivity and objectivity. Each word is given a subjectivity score by experts. The lexicon is used to score the subjectivity of text. A popular lexicon is the TextBlob lexicon. (ADD LINK)

**Machine learning** approaches build algorithmic models which can classify subjectivity in text. Models are trained on a large corpus of pre-labelled fact/opinion orientated text. A popular corpus comes from the researchers Pang and Lee (2004). The corpus consists of 5000 subjective movie reviews from Rotten Tomatoes and 5000 objective movie reviews from IMDb. **(ADD THE LINKS)**

I used the TextBlob implementation as it was one of the only open source implementations available.

**How do you build a subjectivity lexicon and score the subjectiveness of text?**

Building a subjectivity lexicon is very similar to the method for building a sentiment lexicon (see “Building a sentiment lexicon” on the Sentiment Analysis page).

With a lexicon built, subjectivity scores can be calculated for text. This may differ depending on implementation. Since I used TextBlob I will explain how this score is calculated. TextBlob uses simple averaging along with other special case rules to express subjectivity. This calculation differs slightly from the TextBlob sentiment calculation (see “Scoring the sentiment of text” on the Sentiment Analysis page). Let’s break it down.

1. **The base case - simple averaging**

Assume the word “good” has a subjectivity score 0.6. If all other words have a subjectivity score of 0, the sentence “My jail is good” will use the average subjectivity i.e. 0.6/2 = 0.3. We divide by 2 as TextBlob **ignores 1 letter words and words not in the lexicon** (“my” and “is” are not in the lexicon but “jail” is in the lexicon).

1. **Special cases - Negation, modifiers, 1 letter words and unknown words**

If we change the sentence to “My jail is not good” the sentence subjectivity does not change. Therefore, unlike sentiment with TextBlob we **do not** need to apply a **negation multiplier**. Therefore, the sentence’s subjectivity is still given by 0.6/2 = 0.3.

If we change the sentence to “My jail is very good” the sentence becomes more subjective. The word “very” **modifies** the subjectivity of the word “good”. TextBlob’s lexicon not only captures the subjectivity score, but also the intensity score. The intensity score determines if a word modifies the next word. TextBlob recognises “very” as a modifier word and ignores it’s subjectivity score (0.3) and uses it’s intensity score (1.3) to multiply the following word. The sentence’s subjectivity is given by [1.3\*0.3]/2 = 0.39.

If we change the sentence to “My jail is not very good” we have a **negation combining with a modifier**. In this instance, the subjectivity of the word “good” is still not multiplied by a negation multiplier but is multiplied by the inverse intensity (1/intensity score). The sentence’s polarity is given by [(1/1.3)\*0.3)]/2 = 0.23.

**Limitations of subjectivity with lexicons**

Subjectivity lexicons use individual words to determine subjectivity and not the context. Thus, two sentences using objective words would be deemed objective despite potentially being subjective. For example, the sentences “The queen is a monarch” and “The queen is a lizard” would both be deemed objective despite the second being subjective.

**References**

Textblob: <https://github.com/sloria/TextBlob/blob/eb08c120d364e908646731d60b4e4c6c1712ff63/textblob/en/en-sentiment.xml>

Pang and Lee paper: <http://www.cs.cornell.edu/home/llee/papers/cutsent.pdf>

Pang and Lee movie corpus: <http://www.cs.cornell.edu/people/pabo/movie-review-data/>

IMDb: <https://www.imdb.com/>

Rotten Tomatoes: <https://www.rottentomatoes.com/>



**Readability Analysis**

**What is readability analysis?**

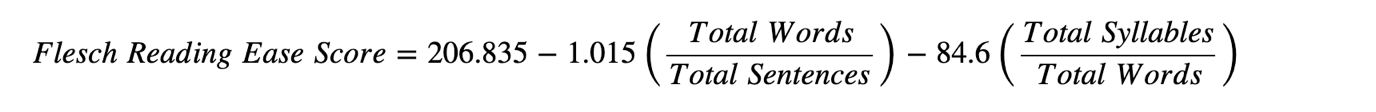
Readability analysisis a natural language processing technique to determine the ease with which text can be read. Popular real-world applications include working out how easy it is to read different types of text, such as, company policies, marketing material and newspaper articles. This approach is useful to help the reader engage with written content.

**Approaches to readability analysis**

The most obvious approach would be to examine the lengths of sentence/paragraph etc. Researchers have developed more sophisticated formulaic methods to calculate readability. These include: Flesch Reading Ease, Flesch-Kincaid Grade Level and Dale-Chall Readability Score. The Flesch Reading Ease is one of the most popular and widely used. This is the approach implemented on this page. ADD LINKS

**How does Flesch Reading Ease work?**

In the 1940s Rudolf Flesch worked on methods to improve the readability of newspapers for Associated Press. He developed the Flesch Reading Ease formula (see Figure 1). The interpretation of scores can be found in Table 1.

Label. The Flesch reading ease score****

Make into a HTML table – score, grade, example

Mathjax for formula in HTML: <http://docs.mathjax.org/en/latest/basic/mathematics.html>

**Graphical user interface, table

Description automatically generated**

90-100 – Understood by the average 10-year-old e.g. comic books

80-90 – Understood by the average 11-year-old e.g. Roald Dahl books

70-80 – Understood by the average 12-year-old e.g. Harry Potter books

60-70 – Understood by the average 13-15-year old e.g. The Guardian articles

50-60 – Understood by 15-18-year olds e.g. non-fiction books

30-50 – Understood by undergraduates e.g. academic papers

0-30 – Understood by university graduates e.g. statutory laws

The readability score is determined by two proportions. The number of words per sentence and the number of syllables per word. The higher these proportions the less readable the text is judged to be.

In the formula the two numbers 1.015 and 84.6 are multiplied by their respective proportions. Given these numbers more weight is given to reducing the readability score for the number of syllables per word than the number of words per sentence.

As a side note, you may have noticed it’s possible to get values outside the range of 100 to 0.

The maximum score is 206.835. This occurs when no text is input. Scores between 100 to 206.835 occur when the number of syllables per word is 1. Scores below 0 can occur if the number of syllables per word is very high.

**Fun fact** – a study has calculated the Flesch Reading Ease score for 2000 articles about people on Wikipedia. The study found articles about sportspeople and entertainers to be the most readable, while articles about scientists and philosophers to be the lease readable. The least readable scientists are economists (Flesch score = 41.7), psychologists (42.25), chemists (42.81) and mathematicians (43.35). (ADD LINK)

**Limitations of Flesch Reading Ease**

The score is based on the proportion of words to sentences and the proportion of syllables to words. This leads to a couple of problems:

1. The formula does not account for other factors which determine the ease of reading. For example:
   * Is the content organised in an effective and ordered way?
   * Does it hold people’s attention?
   * Can people understand it e.g. are you using undefined acronyms.
2. Text must be prepared to avoid misleading results. Embedded punctuation (e.g. a question mark in the middle of a sentence) and text not in full sentences (e.g. headings and bulleted lists) need to be formatted correctly or they will alter the proportion of words to sentences and therefore the Flesch Reading Ease score.

**References:**

Flesch reading: https://web.archive.org/web/20160712094308/http://www.mang.canterbury.ac.nz/writing\_guide/writing/flesch.shtml

Flesch-Kincaid: <https://apps.dtic.mil/sti/citations/ADA006655>

Dale-Chall: <https://www.worldcat.org/title/readability-revisited-the-new-dale-chall-readability-formula/oclc/32347586>

Flesch reading score formula explained: <https://www.linkedin.com/pulse/hacking-flesch-reading-ease-test-paul-croubalian>

Wikipedia 200 articles study: <https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Vital_Articles/Readability>

