SYNAPSE ML TASK 3

**NLP – Natural Language Processing**

NLP combines computational linguistics—rule-based modeling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to ‘understand’ its full meaning, complete with the speaker or writer’s intent and sentiment.

1. **Preprocessing Techniques**

In NLP, text pre-processing is the first step in the process of building a model.

In the vector space model, each word or term is a dimension. The text/document is represented as a vector in the multi-dimensional space.  
The number of unique words means the number of dimensions. Thus preprocessing is widely used for dimensionality reduction.

The various text pre-processing steps are:

1. Tokenization
2. Lower casing
3. Removing unwanted characters
4. Stop words removal
5. Stemming
6. Lemmatization

**Tokenization**: Splitting the sentence into words.

**Lower casing:** Converting a word to lower case. Same words of different cases are represented as two different words in the vector space model (resulting in more dimensions).

**Removing unwanted characters:** For the machine to be able to clearly distinguish words, it has to be stripped of unwanted characters such as digits, special characters and uncharacteristic whitespace. The data may be sourced from various places and as such data which is not in a constructive format for the machine to understand must be removed. This includes URL’s, fancy fonts and styles applied, and unrecognized characters.

**Stop words removal:** Stop words are very commonly used words (a, an, the, etc.) in the documents whose purpose is generally to connect two separate sentence ideas. These words do not really signify any importance as they do not help in distinguishing two documents. For example, in sentiment analysis, words like ‘good’, ’terrible’ are important words while words like ‘do’, ‘says’ don’t help in distinguishing sentiments.

**Stemming**: It is a process of transforming a word to its root form. When a form of a word is recognized it can make it possible to return search results that otherwise might have been missed. That additional information retrieved is why stemming is integral to search queries and information retrieval. Often, the best results can be attained by using the basic morphological form of the word: the lemma. However, stemming may not always be accurate and the root word obtained may or may not be its lemma. For example, caring may be reduced to car.

**Lemmatization**: Unlike stemming, lemmatization reduces the words to a word existing in the language. Lemmatization takes more time as compared to stemming because it finds meaningful word/ representation. Stemming just needs to get a base word and therefore takes less time. Lemmatization is preferred over Stemming because lemmatization does a morphological analysis of the words, however it requires deep linguistics knowledge in constructing dictionaries to look up the lemma of the word.

1. **Techniques for implementation of summarization of text**

**Extractive Summarization vs Abstractive Summarization**

Abstractive Summarization

Carried out using neural networks, Abstractive summarizers are so-called because they do not select sentences from the originally given text passage to create the summary. Instead, they produce a paraphrasing of the main contents of the given text, using a vocabulary set different from the original document.

Some common models are **GPT-2**, **GPT-3**, **BERT**, **OpenAI**, **GPT**, **T5**.

Extractive Summarization

It is the traditional method whose main objective is to identify the significant sentences of the text and add them to the summary. The summary obtained contains exact sentences from the original text.

Concept behind extractive summarization

**Relevance measure:** It expresses the entire text source as a vector of frequencies of terms in the text. Thus, the size of the vector is the number of distinct terms in the text and each value in vector represents the frequency of one particular word in the entire source. Now, the sentence whose vector is most similar to the (text) source vector is added to the summary and the terms in this sentence are removed from the entire source text. The algorithm is repeated until the number of sentences in the summary is k (where k is a parameter to the algorithm) or there is no more text left in the (last updated) source. Also, Cosine similarity, euclidean distance etc. are some measures used to calculate similarity between two vectors.

Some algorithms for extractive summarization

**TextRank**

The gensim package implements TextRank, an unsupervised algorithm based on weighted-graphs. It is built on top of the popular [PageRank](https://en.wikipedia.org/wiki/PageRank) algorithm that Google used for ranking webpages. TextRank works as follows:

1. Pre-process the text: remove stop words and stem the remaining words.
2. Create a graph where vertices are sentences.
3. Connect every sentence to every other sentence by an edge. The weight of the edge is how similar the two sentences are.
4. Run the PageRank algorithm on the graph.

Pick the vertices(sentences) with the highest PageRank score.

Implementation: [TextRank](https://colab.research.google.com/drive/18oAv_W_vHzFh3XCOWVJ-EAcheZBNOx7n?authuser=3%23scrollTo=TJZ2AX2af-dn)

**LexRank**

LexRank is an algorithm implemented using sumy library.

A sentence which is similar to many other sentences of the text has a high probability of being important. The approach of LexRank is that a particular sentence is recommended by other similar sentences and hence is ranked higher.

Higher the rank, higher is the priority of being included in the summarized text.

Implementation: [LexRank](https://colab.research.google.com/drive/10Yr8ez5o71HhqGdQ_nEQwKm27WdroCCm?authuser=3%23scrollTo=rlqkw29RvP8E)

Working of LexRank algorithm

1. Input to the model
2. Word embeddings
3. Intra-sentence cosine similarity
4. Adjacency matrix
5. Connectivity matrix
6. Eigenvector centrality
7. Output of the model

**Input to the model**

The basic input to the model may be just an article or set of articles.

**Word embeddings**

Representation of words in the vector format is called word embeddings.

Usually word embeddings are computed such that the words that are represented as similar vectors are expected to be similar in meaning. For example, a “cat” and a “dog” representation in vectors are very similar whereas “aeroplane” is completely different from both animals words.

**Intra-sentence cosine similarity**

In LexRank implementation an intra-sentence of sentence is used. It means the average of all word embeddings within a sentence that are used to compare to other sentences. Cosine similarity is a metric, helpful in determining, how similar the data objects are irrespective of their size. In this case, how similar sentences are represented by vectors.

The formula for calculating the cosine similarity is:

**Cos(x, y) = x . y / ||x|| \* ||y||**

**Adjacency Matrix**

It is used to represent the similarities across sentences. It is usually a binary matrix with just information whether the two vertices have an edge between them.

**Connectivity Matrix**

A connectivity matrix is usually a list of which vertex numbers have an edge between them. It represents the number of connections a sentence has with other sentences , thus the importance of the sentence stems from the other sentences ‘recommending’ it.

**Eigenvalue Centrality**

To find out the most important sentences LexRank utilizes eigenvector centrality. The method is called power iteration method.

In the first step each matrix row is multiplied by a 1. Then root of square of results is taken, and this step is taken until this value does not show deviation between subsequent iterations.

**Output**

The sentence with the highest standardized eigenvalue score is assigned the highest importance.

**Latent Semantic Analysis**

Latent Semantic Analysis is a unsupervised learning algorithm that can be used for extractive text summarization.

Latent Semantic Analyzer (LSA) is based on decomposing the data into low dimensional space without any significant loss of information. LSA has the ability to store the semantic of given text while summarizing. Unique words in a document can be thought of as dimensions, and comprehensible sentences made with it form vectors which represent word combination patterns which are recurring in the corpus. The magnitude of the singular value indicates the importance of the pattern in a document. A summary hence requires reduction in dimensions which is done efficiently by LSA.

**Luhn**

Luhn Summarization algorithm’s approach is based on TF-IDF (Term Frequency-Inverse Document Frequency). TF-IDF is a numerical statistic that is intended to reflect how important a word is to a [document](https://en.wikipedia.org/wiki/Document) in a collection or [corpus](https://en.wikipedia.org/wiki/Text_corpus). The tf–idf value increases [proportionally](https://en.wikipedia.org/wiki/Proportionality_(mathematics)) to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. It is useful when very low frequent words as well as highly frequent words (stopwords) are both not significant.

Based on this, sentence scoring is carried out and the high ranking sentences make it to the summary.

1. **Conclusion and Choice of Algorithm**

There are two fundamental approaches to text summarization: extractive and abstractive. The former extracts words and word phrases from the original text to create a summary. The latter learns an internal language representation to generate more human-like summaries, paraphrasing the intent of the original text.

While undoubtedly abstractive summarization is miles ahead in terms of potential due to its simulation of a human like understanding of summarization methods, at present the training period is far too long and costly, and results even at moderate training time of around 100K steps do not provide a perfect answer. The Tensorflow network provided by Google requires around 7000 GPU hours for accurate testing.

Thus, coming to extractive summarization, in order to objectively define the algorithm best suited for the task, a few metrics were used to determine this namely ROUGE-1 and BLEU.

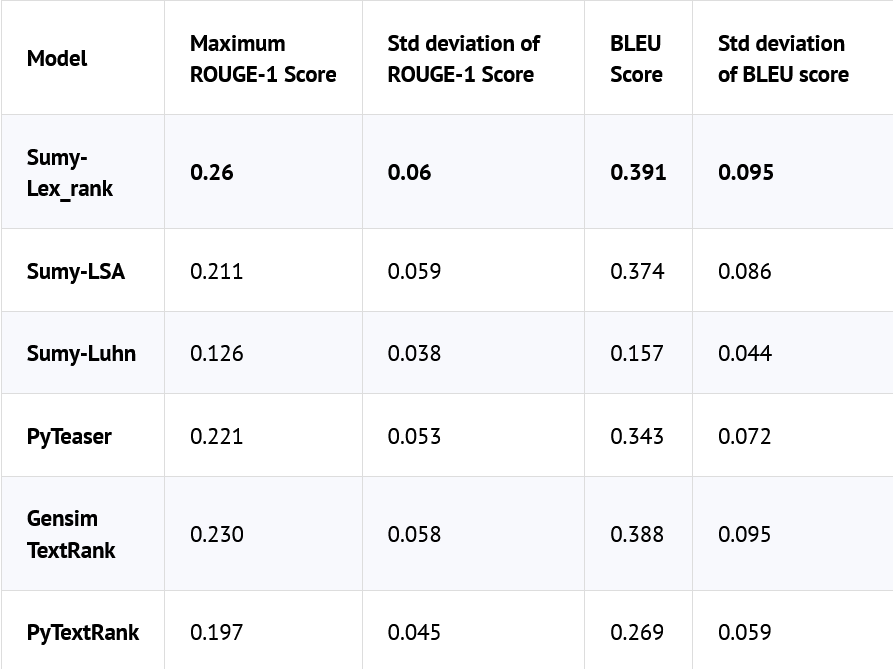
[Rouge-N](http://www.aclweb.org/anthology/W04-1013) is a word N-gram measure between the model and the gold summary. an N-gram is a contiguous sequence of n items from a given [sample](https://en.wikipedia.org/wiki/Sample_(statistics)) of text or speech. The items can be [phonemes](https://en.wikipedia.org/wiki/Phoneme), [syllables](https://en.wikipedia.org/wiki/Syllable), [letters](https://en.wikipedia.org/wiki/Letter_(alphabet)), [words](https://en.wikipedia.org/wiki/Word) or base pairs according to the application. Specifically, it is the ratio of the count of N-gram phrases which occur in both the model and gold summary, to the count of all N-gram phrases that are present in the gold summary. It can also be considered as the recall value measuring how many n-grams from the gold summary translate to the model summary.

BLEU

[BLEU](http://www.aclweb.org/anthology/P02-1040.pdf) metric is a modified form of precision, extensively used in machine translation evaluation.

Precision is the ratio of the number of words that co-occur in both gold and model translation/summary to the number of words in the model summary.

Applying these metrics on various extraction summarization algorithms, LexRank emerges as the one with the highest ROGUE-1 and BLEU score, although TextRank comes close. However **LexRank** is the preferred algorithm for me ultimately, the key difference between the algorithms being the weighting function used for assigning weights to the edges of the graph. While TextRank simply assumes all weights to be unit weights and computes ranks like a typical PageRank execution, LexRank uses degrees of similarity between words and phrases and computes the centrality of the sentences to assign weights. Another point observed from the comparison is that Luhn’s algorithm has a lower BLEU score. This is because it extracts a longer summary and hence covers more reviews of the product.



Data obtained from [here](https://rare-technologies.com/text-summarization-in-python-extractive-vs-abstractive-techniques-revisited/).