Two types of summarization

There are broadly two approaches to automatic text summarization: Extractive and Abstractive.

Extractive Techniques:

"Extractive summarization techniques select relevant phrases from the input document and concatenate them to form sentences. These are very popular in the industry as they are very easy to implement. They use existing natural language phrases and are reasonably accurate. Additionally, since they are unsupervised techniques, they are ridiculously fast. On the other hand, as these techniques only play around with the order of sentences in the document to summarize they do not do as great a job as humans to provide context"

Example summary generated by Extraction Technique for above paragraph:
Extractive approaches use relevant phrases from the input document, then play around with the order of sentences in the document to summarize.

Some Extraction Sumarization Approaches:

NLTK Summarizer

We can start our text summarization journey by trying something simple. So we can turn to the popular NLP package in python — NLTK. The idea here was to summarize by identifying "top" sentences based on word frequency.

Gensim Summarizer

The Gensim summarization module implements TextRank, an unsupervised algorithm based on weighted-graphs from a paper by Mihalcea et al. It is built on top of the popular PageRank algorithm that Google used for ranking.
After pre-processing text this algorithm builds graph with sentences as nodes and chooses the sentences with highest page rank to summarize the document

What is TextRank?
TextRank is based on PageRank algorithm that is used on Google Search Engine. In simple words, it prefers pages which has higher number of pages hitting it. Traditionally, the links between pages are expressed by matrix as shown in the image below. This matrix is then converted to a transition probability matrix by dividing the sum of links in each page which influences the path of the surfer.

TextRank Overview
In the original "TextRank" algorithm the weights of an edge between two sentences is the percentage of words appearing in both of them.

Similarity calculation between sentences
However, the updated algorithm uses Okapi BM25 function to see how similar the sentences are. BM25 / Okapi-BM25 is a ranking function widely used as the state of the art for Information

Retrieval tasks. BM25 is a variation of the TF-IDF model using a probabilistic model.

Improved similarity with BM25
In a nutshell, this function penalizes words that appears in more than half the documents of the collection by giving them a negative value.
The gensim algorithm does a good job at creating both long and short summaries. Another cool feature of gensim is that we can get a list of top keywords chosen by the algorithm. This feature can come in handy for other NLP tasks, where we want to use “TextRank” to select words from a document instead of “Bag of Words” or “TF-IDF”. Gensim also has a well-maintained repository and has an active community which is an added asset to using this algorithm.

Summa summarizer
The summa summarizer is another algorithm which is an improvisation of the gensim algorithm. It also uses TextRank but with optimizations on similarity functions. Like gensim, summa also generates keywords.

Sentence Embeddings
We can use K-means clustering to summarize the types of documents
For clustering the sentences, one has to convert the text to number, which is done using pre-trained word embeddings. We can use GloVe embedding for achieving this. The embeddings created consisted of a dictionary with the most common english words and a 25 dimensional embedding vector for each one of them. The idea is to take each sentence of the document, and calculate it’s embeddings, thus creating a data matrix of (# of documents X 25) dimensions. We can keep it simple for obtaining the sentence embeddings by taking the element-wise weighted average of the word embeddings for all the words in the sentence.
Then, all of the sentences in a document are clustered in k = sqrt(length of document) clusters. Each cluster of sentence embeddings can be interpreted as a set of semantically similar sentences whose meaning can be expressed by just one candidate sentence in the summary. The candidate sentence is chosen to be the sentence whose vector representation is closest to the cluster center.

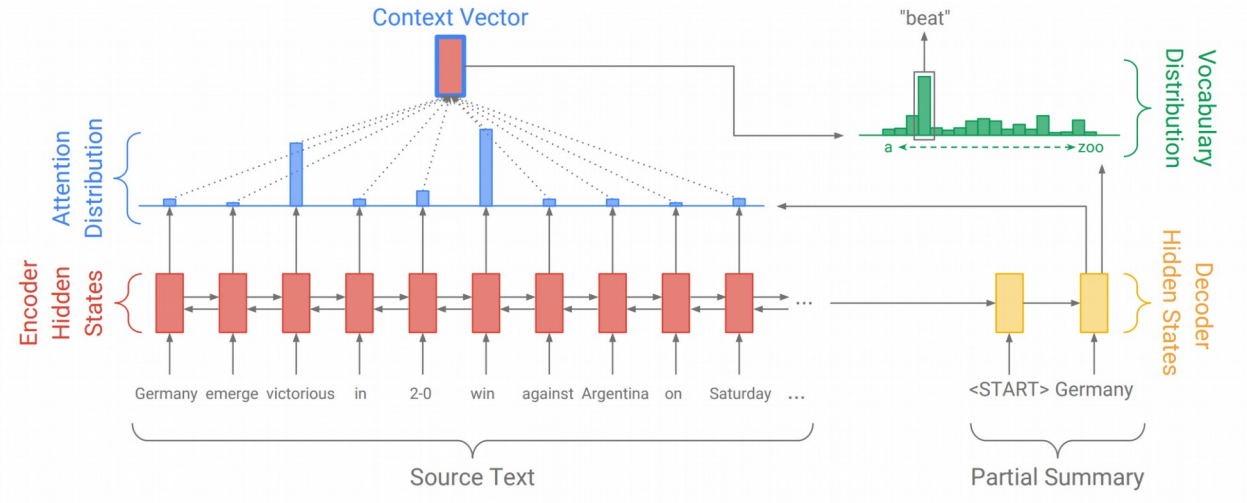
Abstractive Techniques:
“Abstractive summarization reproduces important material in a new way after interpretation and examination of the text using advanced natural language techniques to generate a new shorter text that conveys the most critical information from the original one.”

Abstraction summary approaches:

In the past few years RNNs using encoder — decoder models with attention have become popular for summarization.

Recurrent Neural Network (RNN) – a type of neural network that can perform calculations on

sequential data (e.g. sequences of words) – has become the standard approach for many Natural Language Processing tasks. In particular, the sequence-to-sequence model with attention, illustrated below, has become popular for summarization. Let's step through the diagram!



First, the encoder RNN reads in the source text word-by-word, producing a sequence of Once the encoder has read the entire source text, the decoder RNN begins to output a sequence of words that should form a summary. On each step, the decoder receives as input the previous word of the summary (on the first step, this is a special <START> token which is the signal to begin writing) and uses it to update the decoder hidden state. This is used to calculate the attention distribution, which is a probability distribution over the words in the source text. Intuitively, the attention distribution tells the network where to look to help it produce the next word.encoder hidden states.

Next, the attention distribution is used to produce a weighted sum of the encoder hidden states, known as the context vector. The context vector can be regarded as "what has been read from the source text" on this step of the decoder. Finally, the context vector and the decoder hidden state are used to calculate the vocabulary distribution, which is a probability distribution over all the words in a large fixed vocabulary (typically tens or hundreds of thousands of words). The word with the largest probability is chosen as output, and the decoder moves on to the next step.

The decoder's ability to freely generate words in any order – including words that do not appear in the source text – makes the sequence-to-sequence model a potentially powerful solution to abstractive summarization.