Task 1: we think it is inbuilt in vosk [Vosk Should take care of it, hopefully :)]
Task 2(speaker diarization): not now [**If possible]**
Task 3: Clarify with Abhishek [What do they really mean by regional vocabulary] [**Should cover proper nouns**]
Task 4: Unintelligible content [Not completely understandable, clarify with Abhishek] X
Task 5: Not really sure if this should be here, should be done in the next part. **Just cover what conveys emotions/feelings.**
Task 6: This seems pretty impossible.[**Proper Nouns should be included**]

Jayant:
Models Implemented:

1. **DeepSpeech:**
    a. DeepSpeech is an LSTM based Model, which uses ReLU as activation function for most of its layers.
    b. It was performing well on clear audio, but it did not produce perfect results even on its samples and very very bad results on the noisy audio which we were using for comparing models.
    c. I just need to check it one more time just to be absolutely sure.
2. **Google Speech Recognition API:**
    a. We used the free version of this API, as the paid version was not desired for our project.
    b. The API again was performing well on the sample audio, but was performing very poorly on noisy audio.
    c. Even after denoising our audio, it did not produce very perfect results.
    d. It did add punctuations and full stops at places.
3. **Vosk API:**
    a. Vosk is another free API, which can be used offline too.
    b. Again performed well on clean audio, and gave the most accurate output out of all the models we tried before this, it's output is comparable with Google Speech Recognition API. It was doing good on noisy audio as well
    c. It did miss punctuation marks at places, but wasn't throwing any garbage values when there was no audio.
4. **Wave2Vec2.0 (Facebook):**
    [https://github.com/pytorch/fairseq/tree/master/examples/wav2vec]
    a. Wave2Vec2.0 had a good output for Foreign english audio. It did perform a little poorly on indian english audio. The words were not transcribed properly, (spellings were wrong), words were missing too in indian english audio.
    b. There is a big problem with Wave2Vac, which is a limit to the amount of audio which can be given to it as input at once (I tried giving a 1 min audio as input and my notebook crashed due to all the RAM being consumed, which is a big issue)
    c. Audio will need to be segmented properly with no words cutting for this model to be of use.
    d. The output of Wave2Vec2.0 is in all CAPS which is another issue with that, as we might miss out on acronyms or short-forms due to everything being in capital.

e. It does not add any punctuation to the transcribed text.

Denoisers:

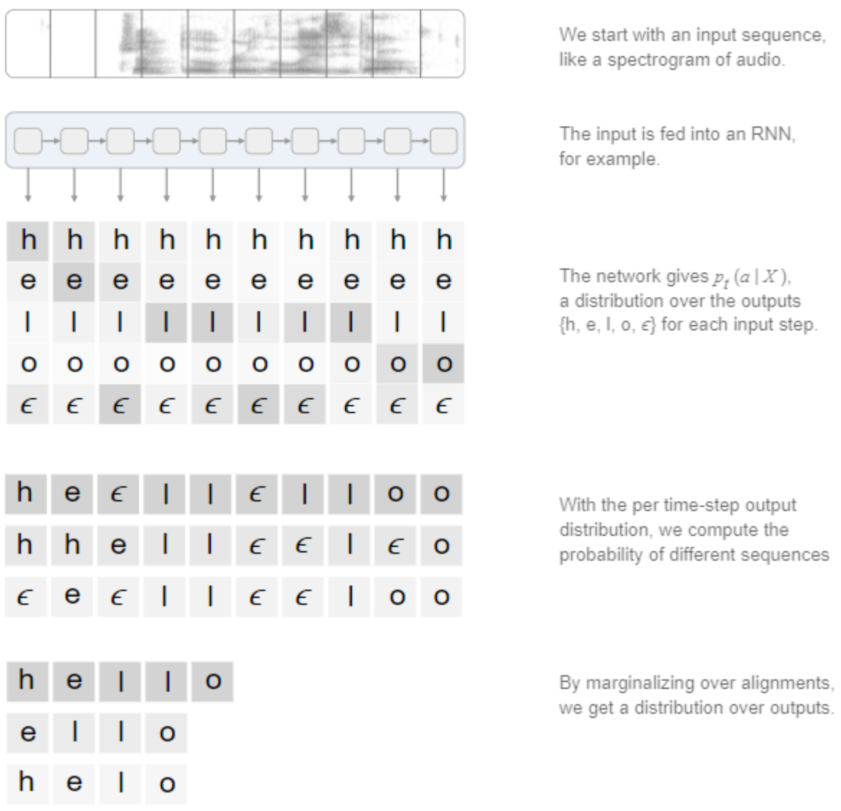
1. Tried Implementing based on CNN which takes spectrograms of different noises as inputs and provides a noise model which can then be subtracted from our noisy audio to provide clean Audio. [https://towardsdatascience.com/speech-enhancement-with-deep-learning-36a1991d3d8d]
    a. It provided a good noise model, but it decreased the intensity of the audio, which degraded the output of the speech to text models.
    b. The noise model is very general and not audio dependent which leads to remnant noise in audio, and does not give a very clean audio.
2. **Facebook Denoiser:** [https://github.com/facebookresearch/denoiser]
    a. It's a very reliable model which cleans the audio to an amazing extent without affecting the intensity of the audio.
    b. Important thing, it does not give out "wav" as output, it gives output in some other format which needed to be converted for input to Google Speech to text API, which is not a problem but needs to be kept in mind.

Sequence Modelling with CTC:

1. **Connectionist temporal classification.**
2. **Does not know the alignment between the input and output.**
3. **For a given input X, it gives us an output distribution over possible Ys, which can be used to assess the probability of a particular output.**
4. **Maximizes the conditional probability [p(Y/X)]**
5. **Alignment :**
    a. **For finding the probability of output given an input, CTC works by summing over the probability of all possible alignments between the two.**
    b. **"Blank Token" introduced in CTC. This corresponds to nothing and is removed from the output.**
    c. **Alignments are of the same length as input.**
    d. **Allowed alignments are those which map to Y (the set of all the characters in the input word) after merging and removing epsilon.**
    e. **For 2 same characters to occur one after the other, they must contain an epsilon between them.**
    f. Allowed Alignments must be monotonic.
    g. Mapping must be many-to-one.
    h. Length of Y cannot be > X.

6. Loss Function:

The CTC alignments give us a natural way to go from probabilities at each time-step to the probability of an output sequence.



We start with an input sequence, like a spectrogram of audio.

The input is fed into an RNN, for example.

The network gives $p_t(a \mid X)$, a distribution over the outputs $\{h, e, l, o, \epsilon\}$ for each input step.

With the per time-step output distribution, we compute the probability of different sequences

By marginalizing over alignments, we get a distribution over outputs.

a. CTC Objective for a single (X, Y) pair is the conditional probability which is the marginalized summation of probability for a single alignment over the set of valid alignments.
b. The final probability is the sum of the final nodes.

The paper is a bit complicated and will require more time investment, so saving that for later :)

<u>Summarization</u> :

- 2 Categories of summarization techniques:
    - **Extractive**: It functions by identifying the important sentences or excerpts from the text and reproducing them verbatim as part of the summary. No New text is generated; only existing text is used in the summarization process

    - **Abstractive**: Interprets text and generates new summary text, as opposed to selecting the most representative existing excepts to perform the summarization.

<u>Extractive Summarization :</u>

Some Basic tasks shared by all summarization techniques:

1. Construct an **intermediate representation** of the input text.
2. **Score the sentences** based on the constructed intermediate representation.
3. Select a summary consisting of the **top k most important sentences**.

[Sentence selection is performed using some specific optimization approach. Sentence scoring determines how well each sentence relays the important aspects of the text being summarized.]

Intermediate Representation:

2 Major categories of intermediate representation:

- **Topic Representation:**
    - Transformation of the text with a focus on text topic identification. Its subcategories are:
        - **Frequency-driven approaches:**
          [Word Probability and TF-IDF] Word frequency approaches.
        - **Topic Word Approaches:**
          There are 2 ways to compute a sentence's importance: By the number of topic signatures, it contains (the number of topics the sentence discusses) or by the proportion of topics the sentence contains v/s the number of topics contained in the text.
          [The first of these tends to reward longer sentences, while the second measures topic word density]
        - **Latent Semantic Analysis (LSA)**
        - **Bayesian topic models - (e.g. Latent Dirichlet Allocation (LDA))**
- **Indicator Representation:**
    - Transformation of each sentence in the text into a list of features of importance; possible features include:
        - Sentence length
        - Sentence position
        - Whether a sentence contains a particular word.
        - Whether a sentence contains a particular phrase.

Using a set of features to represent and rank the text data can be performed using one of 2 overarching indicator representation methods: Graph & Machine Learning Methods.

- **Using Graph Methods:**
    - We find that sub-graphs end up representing topic covered in the text
    - We are able to isolate important sentences in the text, given that these are the ones which would be connected to a greater number of other sentences.
    - We do not need to consider language-specific processing and the same method can be applied to a variety of languages.
    - We can often find that the semantic information gained via graph-exposed sentence similarity enhances summarization performance beyond more simple frequency approaches.
- **Using Machine Learning representations:**
    - The summarization problem is modelled as a classification problem.
    - We require labelled training data to build a classifier to classify sentences as summary or non-summary sentences.
    - To combat the labelled data conundrum, alternatives such as semi-supervised learning show promise.
    - Models which assume dependency between sentences often outperform other techniques.

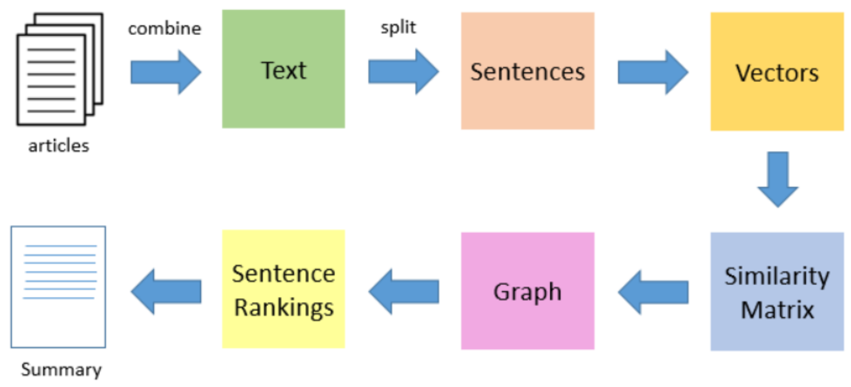
[https://arxiv.org/pdf/1707.02268.pdf]
[https://www.aclweb.org/anthology/W04-1013.pdf]

→ **<u>Text Summarization using the TextRank Algorithm:</u>**

- Text Summarization Approaches
- Understanding the TextRank Algo
- Understanding the Problem Statement
- Implementation of the TextRank Algo
- What's Next?

[Identification of right sentences for summarization is of utmost importance in an extractive method]
[Parts of the summary may not even appear in the original text in abstractive summarization]

- **Understanding the TextRank Algorithm**
  Flow of the TextRank Algorithm :



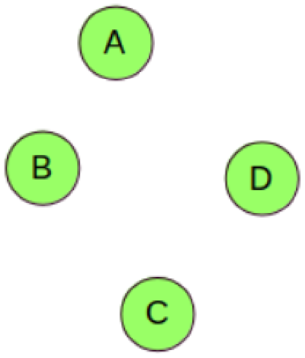
1. The algorithm is pretty straightforward and it joins all the text, takes out sentences from them.
2. The sentences are converted to vectors and then similarity is calculated between the sentences and stored in a similarity matrix.
3. The similarity matrix is converted into a graph and then the sentences are ranked using the PageRank Algorithm (Which calculates the probability of a user to visit one webpage to another in an iterative manner. It is initially filled with probabilities according to links with other webpages).
4. The top 10 ranked sentences are then picked as a summary

→ **Graphs and Graph Theory**

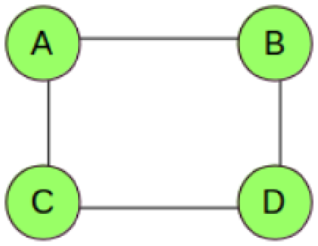
- Some Basic Things:
  - **Vertex** - Point where multiple lines meet. Also known as the node.
  - **Edge** - Line that connects 2 vertices.
  - **Graph** - Combination of Vertices and edges. G = (V, E) ; V = Set of Vertices and E = Set of Edges.
  - **Degree of Vertex** - Number of Edges connected to it.
  - **Parallel Edge** - If 2 vertices are connected by more than 1 edge, then these are called parallel edges.
  - **Multi Graph** - These are the graphs which have parallel edges.

- Graph Terminologies:
  - **Distance between 2 vertices:** Number of edges in the shortest path between 2 vertices.
  - **Eccentricity of a Vertex:** Maximum of distances between a vertex to all other vertices.
  - **Radius of a connected graph:** Minimum eccentricity of all the vertices of a graph.
  - **Diameter of a connected graph:** Maximum eccentricity of all the vertices of a graph.
  - **Central point of a graph:** A vertex for which eccentricity = radius of graph is the central point of the graph.

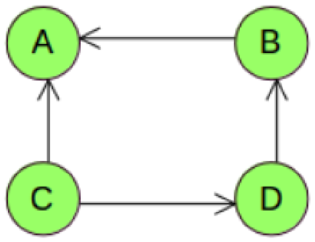
- Types of Graphs
    - **Null Graph:** Graph with no edges.



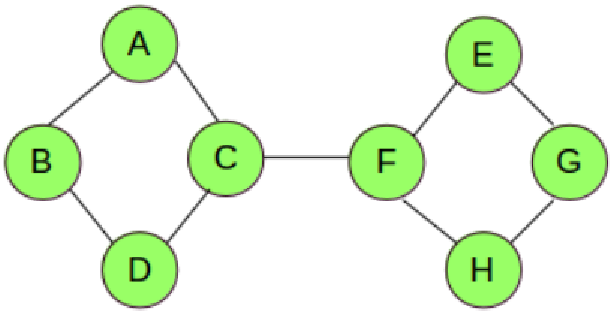
    - **Non-Directed Graph:** Graph with edges but these edges do not have any particular direction.



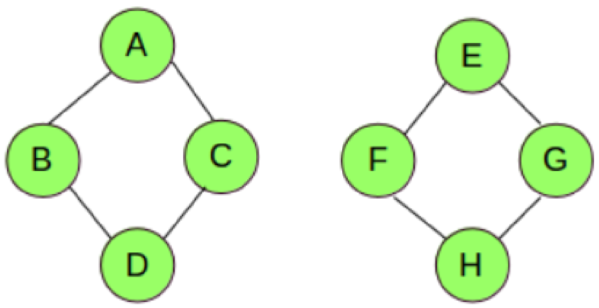
    - **Directed Graph:** When the edges of a graph have a specific direction.



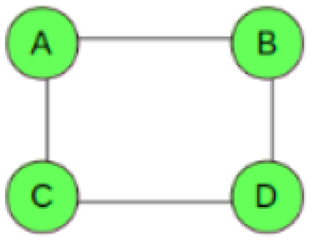
    - **Connected Graph:** When there is no unreachable vertex.



    - **Disconnected Graph:** Graphs which have some unreachable vertex(s).



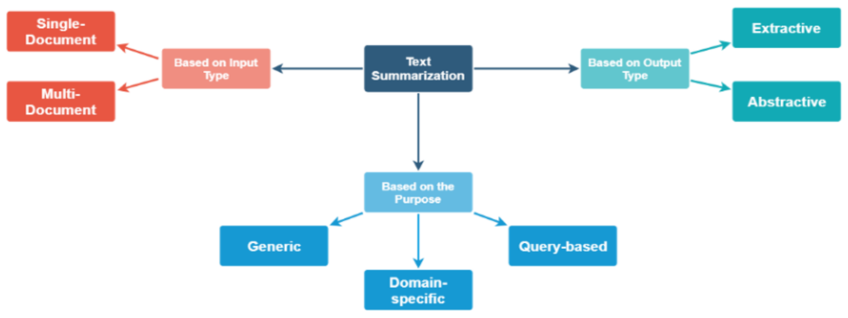
- **Regular Graph:** When all the vertices of a graph have the same degree, these are called k-graphs (where k is the degree of any vertex).



NetworkX is a python package for creation, manipulation and study of the structure, dynamics and functions of complex networks. [It is also an important tool for implementing graphs]

## IMPORTANT ##
Gensim Tutorial :
https://www.geeksforgeeks.org/nlp-gensim-tutorial-complete-guide-for-beginners/

Unsupervised Text Summarization using sentence Embeddings
[https://medium.com/jatana/unsupervised-text-summarization-using-sentence-embeddings-adb15ce83db1]
[https://colab.research.google.com/drive/1pQ6ZsYrkuWsqwjFp87iQGVB0ddkSRRNB]

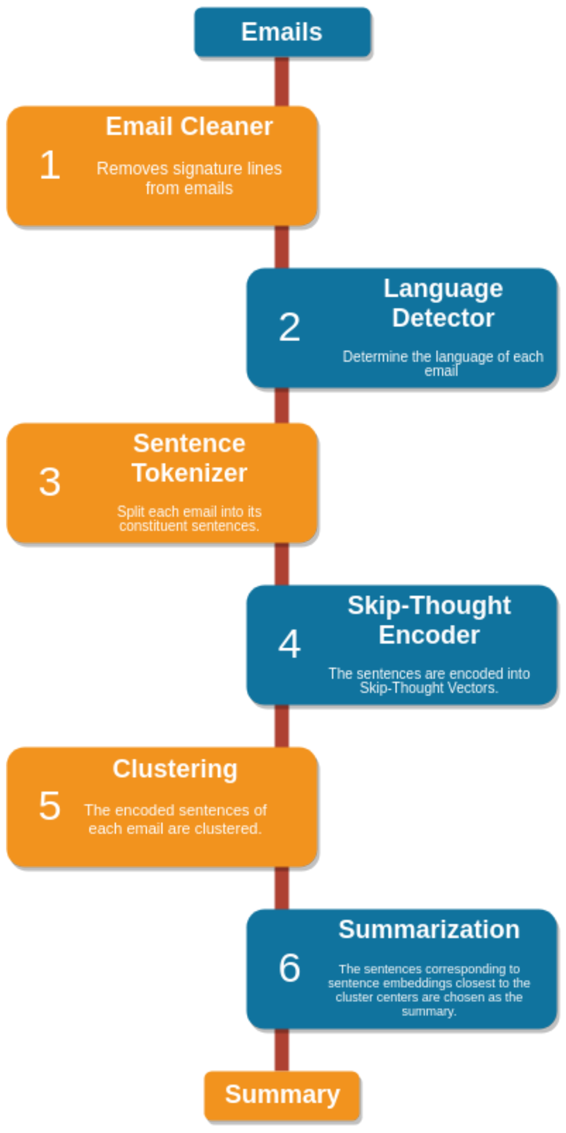


Types of Text Summarization approaches

Purpose specific Summarization

1. **Generic**:
   The Model makes no assumptions about the domain of the content of the text to be summarized and treats all the inputs as homogeneous. The majority of the work that has been done revolves around generic summarization.
2. **Domain-Specific:**
   The model uses domain-specific knowledge to form a more accurate summary. For example, summarizing research papers of a specific domain, biomedical documents etc.
3. **Query-based:**
   The Summary only contains information which answers natural language questions about the input text.

Model Pipeline


Emails
1
Email Cleaner
Removes signature lines from emails
2
Language Detector
Determine the language of each email
3
Sentence Tokenizer
Split each email into its constituent sentences.
4
Skip-Thought Encoder
The sentences are encoded into Skip-Thought Vectors.
5
Clustering
The encoded sentences of each email are clustered.
6
Summarization
The sentences corresponding to sentence embeddings closest to the cluster centers are chosen as the summary.
Summary

Step 1 - Clean the Emails and keep only the required text information.

Step 2 - Detect the language of the mail.

Step 3 - Sentence Tokenization, which can be performed using NLTK's sentence tokenizer.

Step 4 - Skip-Thought Encoder. We need to generate fixed length vector representations for each sentence in our emails. These representations should encode the inherent semantics and the meaning of the corresponding sentence. This can be performed with the help of **Skip-Gram Word2Vec Method**.
Some other approaches.

For Sentence embeddings, one way is to take a weighted sum of the word vectors. Weighted sum is taken because the frequently occurring words ('and', 'to', 'the' etc) provide little information about the sentence. Weights are inversely proportional to occurring frequency.
This method incurs lot of losses as order of words is not taken into account.

Therefore, we choose Skip-thought sentence encoder in a supervised manner to overcome that.
The Skip-Thought model consists of 2 parts:

1. **Encoder Network:**
   It is a GRU-RNN which generates a fixed length vector representation h(i) for each sentence S(i) in the input.
2. **Decoder Network:**
   The decoder network takes this vector representation h(i) as input and tries to generate 2 sentences - S(i - 1) and S(i + 1) which occur before and after the sentence. This is also a GRU-RNN. h(i) acts as the initial hidden state for the GRUs of the decoder networks.

Given a dataset containing a sequence of sentences, the decoder is expected to generate the previous and next sentences, word by word. The Encoder-decoder network is trained to minimize the sentence reconstruction loss. **These learned representations are such that embeddings of semantically similar sentences are closer to each other in vector space, and therefore suitable for clustering.**

Step 5 - After generating sentence embeddings for each sentence in an email, we cluster these embeddings in high-dimensional space into clusters equal to the desired number of sentences. **The desired number is up to us.**

Step 6 - Each cluster has semantically similar sentences and one of them can be used to express the meaning in the summary. The sentence whose vector representation is closest to the cluster center is chosen for the summary. The selected sentences are then ordered to form a summary. Order is determined by the positions of the sentences in their corresponding clusters in the original email. [Sentence is first line if most of the sentences in its cluster occur at the beginning of the email.]

This is a method of extractive summarization.
Tool for Parsing Data from Wikipedia.
Pre-trained Word Vectors
Quick-Thought Vectors, an advancement over the skip-thoughts approach.[Lesser training time and better performance]