**Ananya Hari Narain**

# Automatic Generation of Podcast Summaries from Episode Transcriptions: Spotify Podcast Dataset Challenge

Computer Science Tripos – Part II

Sidney Sussex College

13th May 2022

# Declaration of originality

I, Ananya Hari Narain of Sidney Sussex College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed Ananya Hari Narain

Date 13th May 2022

# Proforma

| | |
|---|---|
| Candidate Number: | **2358G** |
| Project Title: | **Automatic Generation of Podcast Summaries from Episode Transcriptions: Spotify Podcast Dataset Challenge** |
| Examination: | **Computer Science Tripos – Part II, May 2022** |
| Final Word Count: | **11989**[1] |
| Final Code Line Count: | **1917**[2] |
| Project Originator: | Dr Andrew Caines |
| Supervisor: | Dr Andrew Caines & Dr Zheng Yuan |

## Original Aims of the Project

This project seeks to investigate how different summarisation models fare on a on a large corpus of informal and colloquial text data, namely the Spotify Podcast Dataset. This was essentially the crux of the Spotify Podcast Dataset Challenge 2020. It involved training a Bidirectional and Auto-Regressive Transformer (BART) model, implementing a Pointer Generator model and evaluating the model-generated summaries on ROUGE metrics.

## Work Completed

The project has been successful in achieving the success criteria listed in the project proposal. I completed all core tasks and redefined and completed some extensions.
The data was preprocessed in order to prepare for the fine-tuning and I built a Support Vector Machine (using off-the-shelf toolkits) to do so. I also fine-tuned an abstractive summarisation BART model using the dataset that I prepared.
As extensions I fine-tuned Bidirectional Encoder Representations from Transformers(BERT), Text-to-Text Transfer Transformer (T5) and Longformer models to consider their respective performances. I also considered various input filtering methods to vary the way I passed the transcripts to the models to summarise them. I then implemented a Pointer Generator model from scratch, and as a further extension I incorporated a Reinforcement Learning (RL) loss function. The generated summaries were then evaluated on ROUGE metrics as well as METEOR as a metric to compare models that were similiar.

[1] Computed using `texcount`
[2] Computed using cloc : https://github.com/AlDanial/cloc (visited 12th May, 2022)

## Special Difficulties

None.

## Acknowledgements

I would like to extend my thanks to my supervisors: Dr Andrew Caines and Dr Zheng Yuan, whose support and guidance throughout the project was immense. I would also like to thank my Director of Studies, Mr Matthew Ireland, friends and family for their constructive criticism that helped to polish this dissertation.

# Contents

# List of Figures

# Chapter 1

# Introduction

Summarisation is a Natural Language Processing (NLP) task. It involves taking a long text and generating a shortened version that contains the key points from the original.
The goal of the project was to implement an array of summarisation models to summarise podcast episodes from their transcripts, as per the rules of the Spotify Podcast Dataset Challenge 2020. [10]
In the challenge, entrants were given a 13GB data corpus including transcriptions of podcasts and sought to produce the 'best' automatic summarisation models.
I generated summaries that were evaluated quantitatively against automatic metrics as set out by the challenge and qualitatively using volunteers in a human evaluation exercise that I set up.

## 1.1 Motivations

The motivations of this project are twofold: 1) producing summaries to assist the user in deciding which podcast to listen to, and 2) investigating how models in the current NLP literature work on informal datasets.
Firstly, the summaries generated are incredibly useful; many creators fail to provide descriptions of their episodes and ones that are provided vary greatly in quality. Meaningful summaries will give listeners a better feel for the episode content and may impact their listenership. Listeners are more likely to find relevant podcasts if they have access to accurate summaries and hence summaries could be used in automatic recommender systems to enhance user experience. The generation of high-quality summaries is clearly important.
Existing literature pertaining to summarisation uses data from CNN/Dailymail [15] and hence has a focus on formal corpora. My project considers how well-known summarisation models fare against colloquial datasets and is contingent on the quality of the ASR (Automatic Speech Recognition) transcriptions. As speech-to-text is a problem that can be solved well, the methods that I explored can be deemed as useful contributions to the field of summarisation.

## 1.2 Summary of background work

### 1.2.1 Spotify Podcast Dataset Challenge

The challenge that Spotify set was in two parts - one focused on information retrieval and the other on summarisation. I took on the summarisation task. This involved training models capable of generating informative and human-readable summaries from episode transcriptions.

A large data corpus containing relevant information was provided for this purpose.

### 1.2.2 Previous Work

The current state-of-the-art summarisation models are based on Transformer architectures, as proposed by Vaswani et al. [42]. Zhang et al. [44] showed how pre-trained Transformers outperformed many other approaches such as RNNs (Recurrent Neural Networks), on qualitative and quantitative metrics. Pre-trained models only need a limited number of supervised examples to be fine-tuned to specific domains, leading me to consider fine-tuning existing Transformer models such as BERT [26] and BART [23].
Pointer-Generator networks have also produced state-of-the-art results [5] [13]. I implemented a Pointer-Generator network in order to compare their performance with Transformer-based architectures.

As this challenge was open, several papers detailing their approaches were available for perusal. My work is inspired by some of these. Rezapour et al. [35] considered two approaches: generating genre-aware abstractive summaries as well as generating summaries that maximised the number of named entities that appeared in them. They found that the summaries varied from genre to genre. For example, sports summaries included more named entities and true crime summaries included more suspense; 1.1 illustrates these findings.

| | |
|---|---|
| **Sports** | In EP 4 Jon Rothstein is joined by Arizona State Head Coach Bobby Hurley, Creighton Guard Mitchell Ballock, and Merrimack Head Coach Joey Gallo checks in on the Hustle Mania Hotline. |
| **True Crime** | Sometimes, it takes years to connect a killers crimes. On March 6th 1959 a man was born who would, eventually, be tried for the murder of a single woman. It would take years for police to connect him to 4 other murders. Years and a clever investigator who got the DNA he needed. |
| **Comedy** | This weeks episode is a little bit of a throwback.. We have one of our close high school friends Zak on the show, and he is just a blast! This episode is pretty chill, so you better have a drink for this one! |

Figure 1.1: Sample human-generated summaries from different genres.

Rezapour et al. used ROUGE [25] metrics and human volunteers to evaluate the summaries. Manakul et al. [28] investigated the effects of changing the way that input was passed to the models. I explored several input filtering methods considered by Mankul et al. with different Transformer based architectures and a Pointer-Generator network.

# Chapter 2

# Preparation

## 2.1 Starting Point

**Code:** The implementation of the Pointer Generator model was inspired by the one detailed by See et al. [38].
I modified an existing implementation of a seq-2-seq model given in `https://github.com/ymfa/seq2seq-summarizer/blob/master/model.py` and all other code in this project was developed from scratch using well-known libraries listed in 2.4.3.
**Other knowledge and experience:** I have experience programming in Python and used Pandas [29], NLTK [4] and PyTorch [31] for personal projects.

## 2.2 Background Knowledge

This section elaborates on content that is beyond what is covered on Artificial Intelligence and Neural Networks [11] covered in Part 1B of the Computer Science Tripos.

### 2.2.1 Approaches to summarisation

There are two main approaches to summarisation: extractive and abstractive as described in [19]. I consider both of these in my project.
**Extractive** approaches lift critical sections of text directly from the input document, crop and piece them together to produce coherent summaries. The quality of extractive summaries is directly dependent on the quality of the original text and as a result, generated summaries tend to have good grammatical structures.
Madhuri et al [27] describes the extractive framework model as follows:

- create intermediate representations of the input text
- scores phrases based on some metric
- select the n that are deemed most important

**Abstractive summarisation** instead focuses on generating new shorter text segments given the input text. The key difference here is that text in the generated summaries may not have appeared in the original document. Text is rephrased using NLP techniques so resulting summaries may be richer and more complex. These may not be as human-readable as it can be difficult for the model to piece together phrases in a grammatical way.

### 2.2.2 Support Vector Machines

**Support Vector Machines** (SVMs) [7] are supervised learning models that are commonly used for classification and regression. In this project I used an SVM for the classification of transcripts containing adverts.
Given labelled data, each feature vector in the training set of the data is represented as a point in $n$-dimensional feature space. Points are then divided using hyperplanes by finding the hyperplane that divides the categories by the largest margin. The output of the SVM is a set of points specifying these hyperplanes. Classification is handled by predicting where points from the test set are mapped to in the feature space in relation to their proximity to the defined hyperplanes.
Advert classification is a binary problem; transcripts either contain adverts or not so a linear SVM with a Radial Basis Function (RBF) kernel could be used. A kernel function computes the mapping of data points to vectors in the $n$-dimensional feature space. This is defined as:

$$K(x, x') = exp(-\frac{(||x - x'||)^2}{2\sigma^2})$$

where $||x - x'||^2$ is the squared Euclidean distance between two feature vectors.
SVMs have been used historically for summarisation as displayed by the work by Hirao et al. [16] where they employed an SVM to rank sentences as important or not important. Fuentes et al. [12] built on this method to consider multi-document summarisation. Since the advent of Transformer models such as BERT and BART, however, pre-neural methods such as SVMs are not as commonly used for NLP tasks.

### 2.2.3 Background of Transformer models

#### Sequence modelling

Sequence modelling refers to how we predict the next word to output in a sequence. With summarisation, summaries are the output of a sequence model. Standard NNs cannot be used to model sequences they process sequences as a single item but in order for us to build up contexts of words we must process sequences one word at a time.
Recurrent-Neural-Networks (RNNs) [36] can consider the contextual meanings of words within a sequence. Contexts are updated based on the current context and the word, for each word in the sequence. The context for 'bank', for example, is different in the sentence 'She withdrew money from the bank' and 'She sat by the river bank'.

#### Seq-2-Seq models

Seq-2-Seq models [41] are NNs that take an input sequence and output a different sequence. The input sequence is passed through an encoder-RNN which produces an context vector containing the sentence level representation of the input text sequence. The encoder-RNN consists of an attention layer 2.2.3 and a feed-forward network. Layer normalisation [1] is applied to these components to normalise their distributions; this may involve rescaling activation functions and adding biases.

A decoder-RNN outputs a sequence of words to form a summary and takes the input of the previous word of the summary to update the decoder hidden state. RNNs, however, cannot consider relations between words that are far apart. LSTM (Long-Short-Term-Memory) [17] is

an example of a recurrent unit that has been proposed to tackle long-range word dependencies. 2.1 gives an overview of basic seq-2-seq architecture.

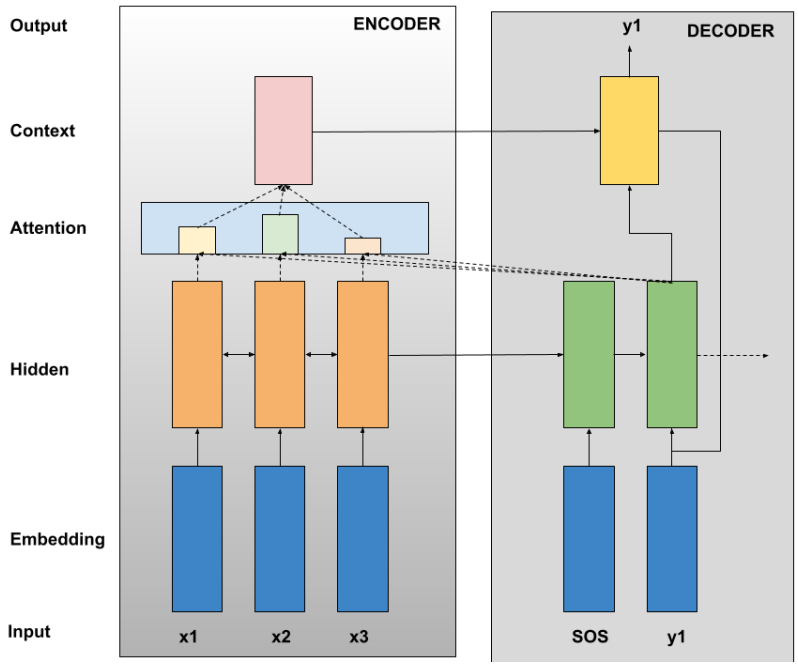


Figure 2.1: Seq-2-seq architecture.

### Attention

Attention mechanisms relate parts of a sequence that are separated by some distance. For example:
X spoke German as they were born in Berlin.

Bahdanau and Self Attention are the two types of Attention mechanisms. Bahdanau Attention [2] considers a weighted sum of previous context vectors. Self-attention considers the whole input sequence at once and transforms it into another sequence containing information on the relatedness between words in the sequence to the others.

## 2.2.4 Transformers

Transformer architectures were introduced by Vaswani et al. [42] who focussed on attention mechanisms over recurrency. Transformers can train a model in fewer steps, making them the current start-of-the art models in summarisation.

### Self-Attention

Self-attention is the key to Transformer architectures. It is an operation that transforms a sequence of vectors $(x_1, x_2, \ldots x_n)$ into another sequence of vectors $(y_1, y_2, \ldots, y_n)$. Weights $w_{ij}$ are derived from the input vectors $x_i, x_j$ as per the following equations:

$$w'_{ij} = x_i^T . x_j \quad , \quad w_{i,j} = \frac{exp(w'_{ij})}{\sum_j exp(w'_{ij})} \quad , \quad y_i = \sum_j w_{ij} . x_j \tag{2.1}$$

This mechanism calculates the relatedness between each word and every other word by calculating the dot-product. Words that have high attention scores are are more related to other words and hence more important to the overall summary. The RNN-decoder takes attention

as an input, giving it a better sense of which word to predict next.
Self-attention also overcomes the issue with RNNs in that it can consider relations between words that are far apart. Transformer architectures are heralded as a major milestone in the field of NLP as they can accurately model inter-word relationships.

### 2.2.5 BERT

BERT is a Transformer-based model proposed by Devlin et al. [9]. Unlike the unidirectional encoders in Transformers, BERT includes a bidirectional encoder 2.2 meaning that input sequences are read bidirectionally. This ensures that there is less reliance on reading from left-to-right which may bias words towards certain meanings as the sentence grows. Self-attention is used to decide which words to attend to most to create a context vector.

BERT is pre-trained on two tasks using this bidirectional capability: Masked-Language-Modelling and Next-Sentence-Prediction.
In Masked-Language-Modelling, 15% of words in each sequence are replaced with `MASK` tokens and the model learns the masked tokens based on the context.
In Next-Sentence-Prediction, BERT considers pairs of sentences such that it learns to predict whether the first sentence precedes the second.
BERT [26] may be used for extractive summarisations; Liu [26] fine-tuned BERT to produce high quality summaries.

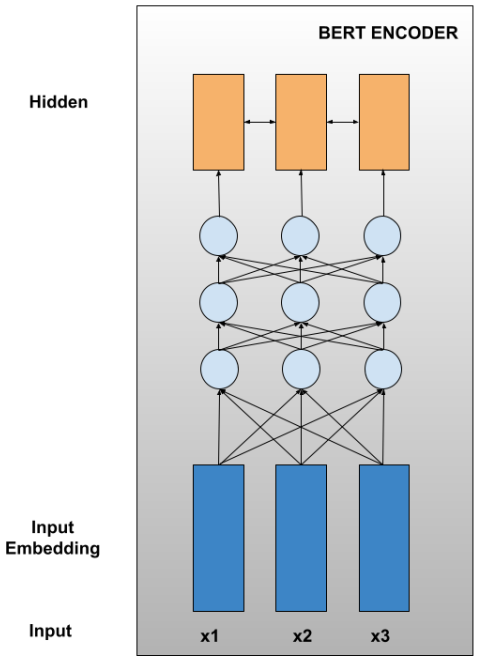


Figure 2.2: BERT: Bidirectional encoder.

### 2.2.6 BART

BART is a Transformer-based model proposed by Lewis et al. [23]. BART uses bidirectional encoder (much like BERT) and left-to-right autoregressive decoder. The autoregressive decoder looks at previous words in the sequence in order to predict the next word in the sequence.

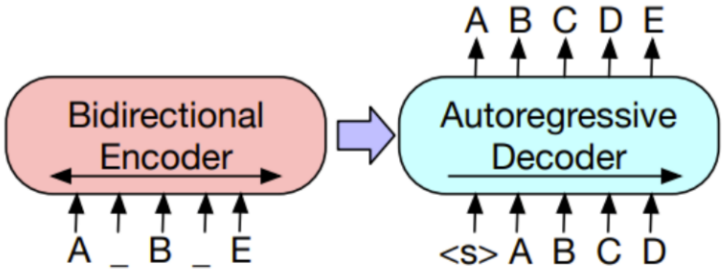


Figure 2.3: Encoder-Decoder structure of BART (from Lewis et al.).

BART is pre-trained with a range of different measures in order to add noise. These include:

- random shuffling of the original text
- token masking, whereby random spans of text are replaced with a single mask token
- token deletion, whereby random tokens are deleted from the original text so the model learns to predict the token context

BART models may be used as examples of abstractive summarisation models, as exhibited by Guo et al. [14].

### 2.2.7 T5

Text-to-Text-Transfer-Transformer (T5) [34] models are Transformer-based models. The differences from traditional Transformers include:

- A simplified layer normalisation. Activation functions are only rescaled and there are no additive biases applied.
- Relative position embeddings are used, as opposed to fixed ones. Positional embeddings are needed to inform the model about where tokens are located in the context of a full sequence. Fixed encodings limit the number of tokens that models can process whereas relative positional encodings generalise sequences of unseen lengths as it only encodes relative pairwise distances between tokens.

### 2.2.8 Pointer Generator (PG) Model

Pointer Generators are hybrid networks that combine the abstractive and extractive summarisation methods. It can choose to copy words from the original text by 'pointing' as well as generating words from the fixed vocabulary. 2.4 illustrates how the pointer mechanism is combined with the generator network.

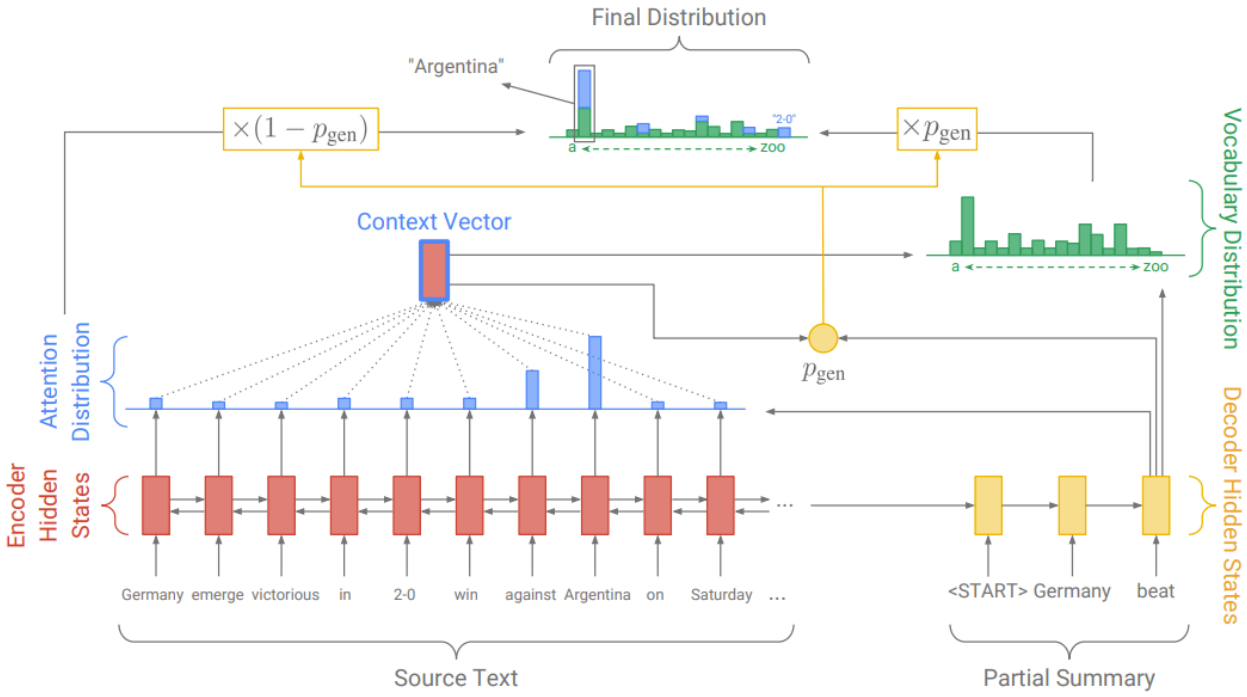


Figure 2.4: Architecture of a Pointer Generator model (from See et al.).

Attention and vocabulary distributions are calculated. We also calculate a generation probability $p_{gen}$. This is the probability of generating a word from the fixed vocabulary; the probability of pointing to a word from the original text is $(1 - p_{gen})$. The probability of the final distribution is:

$$P_{final}(w) = p_{gen}P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i$$

2.4 shows a diagrammatic view of this formula.

### Advantages of PG models

Summaries produced by seq-2-seq models may contain inaccurate factual details if we have rare out-of-vocabulary (OOV) words. An example may be 'The Pistons lost 108-114 to the Bulls, who put in a great performance in the latter half'. Post-tokenisation the numerals in '108-114' will be replaced by generic numbers, are unlikely to have appeared in the training set and are likely to be considered as OOV words. Words are mapped to their embeddings during computation and without an embedding there is no notion of context. This sample transcript could be summarised as 'The Pistons defeated the Bulls 108-114'. A PG model would seek to minimise these types of errors by using the pointer mechanism to correctly reproduce snippets of the original source text.

Seq-2-seq summaries could also contain repetitions in the summary. An example of this may be 'Germany beat Germany beat beat ...'. This could be due to a heavy reliance on the decoder input, leading to repetitive cycles. PG's pointer and coverage mechanisms will tackle this issue. Coverage mechanisms keep track of the amount of attention each word has received so far, and coverage loss functions will penalise attending to already summarised words. The coverage

vector is defined as follows where $a^i$ describes the encoder attention weights:

$$c^t = \sum_{t'=0}^{t-1} a^{t'}$$

Coverage loss is defined as follows:

$$covloss_t = \sum_i min(a_i^t, c_i^t)$$

## 2.3 Evaluation

### 2.3.1 ROUGE

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [25] describes a set of metrics for evaluation of automatic summarisation of texts and machine translations.
Generated summaries are compared against a reference. Some of the more common metrics used to evaluate reference and generated system summaries include ROUGE-1F, ROUGE-2F and ROUGE-LF.
ROUGE-N recall is calculated by the number of matching n-grams/ the total number of words in the reference summary.
ROUGE-N precision is calculated by the number of matching n-grams/ the total of words in the system summary. ROUGE-NF is the F1 measure for n-grams, which is the harmonic mean of precision and recall:

$$F1 = \frac{2}{Recall^{-1} + Precision^{-1}}$$

ROUGE-1 is the unigram model, ROUGE-2 is the bigram model.
ROUGE-L measures the longest matching sequence of words using a LCS algorithm. The precision is calculated by the LCS score of the reference and system summaries/ length of the reference summary.
The recall is calculated by the LCS score of the reference and system summaries/ candidate summary sentence.
The ROUGE-LF is calculated in the same way that the F measures are.
If we consider the following example:

| | |
|---|---|
| System summary | The phone was found under the bed. |
| Reference summary | The phone was under the bed. |

The ROUGE scores will be:

| | |
|---|---|
| ROUGE-1F | 0.92 |
| ROUGE-2F | 0.73 |
| ROUGE-LF | 0.92 |

The scores that were calculated were the averages for all system and reference summaries. In my project I chose to consider all these metrics; for each model I reported 3 different metrics:

ROUGE-1F, ROUGE-2F and ROUGE-LF. Most of the existing literature on summarisation reports F1 scores only, as this places equal importance on how similar the candidate summary is to the reference summary and vice versa; these are proportional to the similarity metrics that are used, i.e. LCS or number of common n-grams. However, when scores are averaged, these scores are less easy to interpret. F1 scores tend to be considered as a measure that describes how similar the candidate summaries are to the reference summaries.

### 2.3.2 Perplexity

Perplexity is a measurement of how well a probability model can predict a sample. The lower the perplexity score, the better the indication of how well the probability model can predict a sample, for this reason perplexity is known as the weighted average branching factor. It is calculated as follows:

$$PP(W) = 2^{-\frac{1}{N}\log_2 P(w1,w_2,\ldots,w_N)}$$

For example, if we have a training set of 10 words, each occurring independently with an equal probability of 0.1, the perplexity of the language is 10. Natural languages rarely follow a unigram model; we have words that have a higher or lower surprisal based on the preceding words in the sentence. Hence I considered bigram perplexity as an evaluation metric.

### 2.3.3 Macro and Micro Precision and Recall

Macro-precision measures the average precision per class. This is calculated as follows:

$$\frac{P_A + P_B + \ldots + P_N}{N}$$

With this metric, all classes contribute equally regardless of how often they appear within the dataset.
Micro-precision measures the precision of the aggregated contributions of all classes. This is calculated as follows:

$$\frac{TP_{sum}}{TP_{sum} + FP_{sum}}$$

The micro precision values may be high even if the model performs poorly on a rare class as it gives a greater weight to the common classes.
Macro-recall measures the average recall per class. This is calculated as follows:

$$\frac{R_A + R_B + \ldots + R_N}{N}$$

Micro-recall measures the recall of the aggregated contributions of all classes. This is calculated as follows:

$$\frac{TP_{sum}}{TP_{sum} + FN_{sum}}$$

Like the micro precision metric, the micro-recall metric weights the recall of the common class more highly than that of the rare classes.

## 2.4 Project Management

In this section I provide the rationale behind the choices that I made regarding tools, language, libraries, corpora and GPU-provider for this project.

### 2.4.1 Tools utilised

**Jupyter Notebooks**: These are web applications that assist in the creation and sharing of code files. Colab Notebooks were used to run the project; these are Jupyter Notebooks that are hosted on Google's server with access to GPUs. Colab Pro is Google Colab's paid subscription model which gives users access to faster GPUs and longer runtimes.
**Git**: This is a version control software that was used for versioning as well as code backups. Any work that I carried out on my local machine that did not take place via Google Colab was pushed to GitHub in the same way and also backed up on OneDrive automatically. In this way all of my code files were kept on two different cloud services. Every month I backed up all my progress onto an external HDD to further protect against corruption or failures.

### 2.4.2 Languages

The programming language that I adopted for the project was Python. This was due to the amount of experience I have in this language as well its support for creating, training and deploying machine learning models.

### 2.4.3 Libraries

The main Python libraries I used in this project are as follows:

| | |
|---|---|
| Pandas [29] | Used in the preprocessing stages when I was handling, analysing and cleaning large volumes of input data. |
| PyTorch [31] | Used for fine-tuning and implementing models. |
| NLTK [4] | Used to tokenise all input text that was being passed to the models for training. |
| Rouge[1] | This was a Python API that reported the ROUGE metric from the paper [25]. |
| Hugging Face Transformers [43] | Used for gaining access to Hugging Face's pre-trained models. |

### 2.4.4 Dataset

The dataset that was used was the one provided by Spotify [6] which contained episode transcripts, episode names, episode IDs, creator descriptions and timestamp data. In order to obtain this, permission had to be obtained by filling out a form on the official Challenge website. This was used for training, validating and testing all the models that I fine-tuned or implemented.

The transcript is provided, as well as a 'confidence' metric which tells the likelihood of the transcription being accurate and faithful to the original audio file. This metric was necessary as the transcript was obtained via Automatic-Speech-Recognition (ASR).

**Snippet of transcript**


```
{"results": [{"alternatives": [{"transcript": "Hi and welcome to the NC
DPS safety scoop a podcast sponsored by the North Carolina Department of
Public Safety.", "confidence": 0.8450406193733215, "words":
[{"startTime": "0.900s", "endTime": "1.100s", "word": "Hi"},
{"startTime": "1.100s", "endTime": "1.200s", "word": "and"},
```

```
{"startTime": "1.200s", "endTime": "1.500s", "word": "welcome"}
... ]}]}]}
```

The BART model that I used was pre-trained on the CNN/Dailymail Dataset [15]. The T5 model I used was pre-trained on C4; a cleaned version of Common Crawl's web crawl corpus. The Corpus of Contemporary American English (COCA) dataset [8] is the largest structured corpus of American English consisting of fiction, magazines, newspapers, and academic texts. I trained a simple Laplace model with this corpus to consider word occurrences and evaluated the summaries of the models against this by writing code to implement the formula for perplexity as aforementioned.

### 2.4.5 GPU

Due to the volume of data that was to be processed and the size of the models that were to be fine-tuned or implemented, a GPU was used. For this I primarily used Google Colab which gave me access to NVIDIA TESLA-K80 GPUs for up to 12 hours at a time. When training the PG model, I upgraded to Colab Pro which gave me access to longer runtimes and T4 or P100 GPUs.

### 2.4.6 Functional requirements

As outlined in my proposal, the objective of my main project was to achieve some core tasks which are as follows and explained in further detail in the Appendix A:

1. Preprocessing of the Spotify corpus. This involved processing and cleaning the dataset and removing transcripts that contained references to sponsorships using a SVM.

2. Fine-tuning a pre-trained BART model on the modified Spotify dataset and experimenting with various filtering methods in order to vary the way the model accepted the input.

3. Implementing a Pointer Generator model as per See et al. [38].

4. Evaluation of the models using ROUGE metrics and human volunteers.

I also planned on completing some extensions, if time permitted; there were some ideas that I had detailed in my proposal. To summarise, these extensions were to: 1) implement a neural network for the preprocessor, 2) consider ways I could handle multimodal analysis, 3) focus on category aware summarisation, 4) train a model using reinforcement learning.

Over the course of the project I realised that some of these extensions were infeasible given the time constraints, hence I considered others. These were as follows:

- Fine-tuning a Bidirectional Encoder Representations from Transformers (BERT) [9] model to obtain extractive summaries.

- Fine-tuning a Text-to-Text Transfer Transformer (T5) [34] model. This was inspired by researching some other competition entries and seeing its performance in relation to BART.

- Implementing reinforcement learning loss functions into my Pointer Generator model following the work of Paulus et al (2017) [32].

- Investigating the effects of capturing long-form content structure with Hugging Face's Longformer [3] models, which can process much longer inputs than BART and BERT.
- Considering other metrics for evaluation such as perplexity and METEOR, detailed respectively in 2.3.2 and 4.3.2.

### 2.4.7 Software Engineering Approach

For my project I took ideas from the Agile methodology. I broke down the requirements into small independent tasks had milestones to indicate their successful completion. Tasks were timetabled into two-week chunks and I used a Notion board to track these tasks, progress, milestone lists and any bugs that needed fixing.
This approach enabled me to make changes quickly. For example, I initially intended to complete some extensions that proved infeasible. After making no progress within the scheduled two-week period, I was able to rethink my ideas and implement extensions that were more feasible.

# Chapter 3

# Implementation

## 3.1 Data preprocessing

This section describes the steps followed in processing the dataset in order to mould it into a form that would be compatible with the various models that I wanted to train and implement.

### 3.1.1 Data extraction

The dataset used for this project, as aforementioned, was the Spotify Podcast Dataset [6]. The transcript data was stored in 3 `.tar` files, each displaying the following structure:

```
podcasts-transcripts -> spotify-podcasts-2020 -> podcasts-transcripts ->
3;4;5 -> 0;1;2;3;4;5;6;...;Y;Z -> show_{show_url} -> {episode_uri}.txt
```

Episode transcriptions were also stored in segments within the file and were not contiguous. Details on the corresponding creator descriptions were stored in `metadata.tsv` under the field name `episode_description`. I concatenated transcript and `episode_description` information into one cohesive file.
Due to the computational resources and time available, I constructed a training dataset by randomly selecting 10,000 (transcript, creator description) pairs from the given 105,360. This process took roughly 8 hours to complete. Test data was procured by randomly selecting 600 of the provided 1027 (transcript, creator description) pairs for testing.
I also cleaned the dataset to remove any transcripts or creator description summaries containing adverts or sponsorships within them. Sponsorship content is rarely relevant to the actual content of the episode. To not include sponsorship related information in the podcast summary is to enhance its quality, especially as the lengths of the generated summaries are size restricted. I employed two different approaches to clean the dataset of adverts. I constructed a simple regex machine to detect and flag up common phrases in ads and trained an SVM with manual annotation of the constructed dataset.

### 3.1.2 Regex machine

I instantiated a database to write the table to, concatenated the transcript chunks to form a contiguous transcript chunk, and checked if the dataset contained sponsorships, urls or had a length less than 100 words. By doing this I could also eliminate transcripts that were too short and were unlikely to generate good summaries due to very minimal information content.

Examples of phrases I included in the sponsorship regex list were: 'This podcast is sponsored by *', 'Before we start', 'Anchor' etc. 'Anchor' is a company that I observed that was prevalent

in all of the adverts that I inspected. I then wrote these filtered podcasts into a database with the fields: `episode_id`, transcript. The advantage of this approach was that it did not involve annotation of the dataset. Due to the sheer volume of data to process this took 6.4 hours to complete.

### 3.1.3 SVM

SVMs are supervised classification models and annotated training data needed to be provided as input. This involved me undergoing many hours of manually annotating a subset of data with Boolean values corresponding to containing an ad and the converse. The process of manual annotation involved reading each transcript to search for ads; most tended to appear at the start of the transcript.
I encountered some ambiguities about what constituted an advert as some podcasts take more covert approaches than others with regards to including sponsorships without properly disclosing them. Some podcasts are also funded by organisations, for example, some podcasts were created by a Californian church, which begs the question: is this classified as an advert or not? I decided for convenience that only explicit mentions of sponsorships or ads would be considered.
These annotations were stored alongside episode transcriptions in another `.csv` file that I later pickled as before to conserve memory.

**Tokenisation**

My approach to tokenisation consisted of the following steps:

- Tokenised each sentence of the text as per the `word_tokenize()` function provided by `NLTK` which splits sentences into tokens.
- Removed standard English stop words. These are the most common words in the language, such as articles, prepositions and pronouns that do not add distinctive information to the text. These stop words were obtained via the `NLTK` library.
- Lemmatised the text as per the `WordNetLemmatizer` class which `NLTK` provides an interface to. Lemmatisation is the process of identifying and removing the inflectional endings of words to return the base of the word, known as a lemma.
  For example, lemmatisation correctly identifies the lemma of'staring' as 'stare'. If we only removed the suffix '-ing' as per the process known as stemming, we would get 'star'.
- Put all the words into lower case.

The SVM was implemented using `scikit-learn`, referred to henceforth as `sklearn` a Python library for machine learning. I created a Pipeline which enabled me to assemble several steps that could be cross validated together. This involved running a CountVectorizer to create a matrix of token counts, a TfidfTransformer over the count matrix to create a normalised tf-idf representation. This meant that I could downweight tokens that occurred very frequently in a corpus.
This weighting is done as follows:

$$w_{x,y} = tf_{x,y} \cdot log(\frac{N}{df_x})$$

, where $tf_{x,y}$ is the frequency of x in corpus y, $df_x$ is the number of documents containing word x and N is the total number of documents. This weighting is described by Spärck Jones (1972) [40] such that terms should be weighted according to their collection frequency.

I used an RBF kernel, one of the simplest ones available, to train the SVM with `gamma`= 'scale', such that

$$gamma = \frac{1}{num\_features \cdot \sigma^2}$$

The `gamma` parameter determines the influence of a single training example on the function; I selected the gamma value to scale such that the Gaussian class boundaries dissipate inversely to the number of features of the model. I considered a 70:20:10 training:validation:test split. In order to evaluate the performance of the SVM I used standard macro and micro precision and recall measurements.

| | |
|---|---|
| Macro precision | 0.952 |
| Micro precision | 0.955 |
| Macro recall | 0.925 |
| Micro recall | 0.955 |

Table 3.1: **Macro** and **micro** precision and recall for SVM performance.

Due to the sheer volume of data to process this took 8.2 hours to run.

### 3.1.4 Hybrid approach

The regex machine performed very well and had precision of 1.0 and recall of 0.99. This approach worked so well because all ads were from Anchor[1].

I hypothesise that Spotify only included transcripts sponsored by Anchor, a Spotify-owned company, in the dataset they released. Anchor is a company that is owned by Spotify.

I ultimately decided on a hybrid approach where I used the regex machine to create a training set and running the SVM on this data to train it and re-ran it on the entire dataset to pick up on any other transcripts containing ads that were not picked up on previously. Some new ones were detected and the dataset was updated to reflect this. Due to the sheer volume of data to process this took 7.8 hours to run.

## 3.2 BERT with k-means clustering (*Extension*)

As an extension I decided to consider the baseline performance to be an abstractive BERT model. I used the pre-trained `bert-base-cased` model and tokeniser, both from the Hugging Face transformers library.

The tokeniser is based on WordPiece [37] that initialises the vocabulary to every character in the training data, merges character pairs and adds the pair that maximises the likelihood of the training data to the vocabulary.

The model is cased, and protects the existing cases of the input words. I chose this model as the Spotify corpus contains named entities[2]. By distinguishing between upper and lowercase words we may be able to distinguish between named and non-named entities and it is often important to have named entities in summaries.

The model was fine-tuned as before, using the same base code as the other Hugging Face models

[1] Verified by running a few simple regexes over the dataset.

[2] Real-world objects, such as people and products.

did, but k-means clustering was applied to all the token embeddings.
The mean pooling of the token embeddings is calculated. The mean pooling is an average value for each group of token embeddings resulting in downsampled token embeddings. This is calculated as below:

$$\frac{\sum_{i}^{len(embeddings)}(embeddings_i \cdot attention\_mask_i)}{\sum_{i}^{len(attention\_mask)} attention\_mask_i}$$

I then created an instance of a KMeans model using the sklearn library. The KMeans model considers `n_clusters`, which indicates the number of the clusters that the model will cluster the input data into. I chose to consider the 'elbow method' to calculate the total number of clusters to consider.
K-means clustering is a method that partitions n observations into k clusters such that each observation is assigned to the cluster with the nearest mean cluster centre, that with the least squared Euclidean distance.

**Elbow Method**

The elbow method is a heuristic that is used for determining the optimal number of clusters in a dataset. Here I plotted the sum of squared distances for k in the range from 1 to the total number of sentence embeddings. The plot looks like an arm with the 'elbow' being the optimal k. The 'elbow' is the cutoff point, where adding one more cluster does not lead to better modelling of the data. In this case the elbow fell near the square root of the total number of sentence embeddings.
Each sentence embedding is then assigned to its own unique cluster, such that we get the minimum distances from each sentence cluster to each sentence in the embedding list. K-means clusters will contain sentences that are deemed as similiar based on the sentence embeddings. Finally I output the sentences in order of their similarity scores in the clusters. These summaries may then have a good sense of the groups of different sentiments and themes within a transcript.

## 3.3 BART baseline

I used an implementation of BART that was available in the Hugging Face Transformers library, namely the `facebook/bart-large-cnn` model that was trained on the large CNN/Daily Mail dataset.
Tokens are sequences of characters that are grouped together as a semantic unit used for processing. Tokens may consist of more than one word or subwords. For example, 'aren't' may be tokenised as 'are not'. The maximum input token length that the model accepted: `MAX_LEN` was set to 512.
The summary length, i.e. the maximum number of tokens that can comprise a summary, `SUMMARY_LEN` was set to 144.

**Batches**

Training and validation took place in batches. The batch size represents the total number of samples that are processed before the model weights are updated. The larger the batch size, the faster the convergence can be, provided that the learning rate is reasonably low. However, due to the memory limits of Colab, I settled on a batch size of 16, which was the largest batch size that the GPU could handle before crashing. Notably, this is a power of 2, as the GPUs' architecture which contains power of 2 execution units.

Due to the amount of time that each training epoch took on Colab, I set the number of epochs to 4.

**Tokenising**

I created a CustomDataset class for tokenisation of the source text (transcript) and the target text (creator description, which was used as the gold standard summary). The tokeniser used was also a pre-trained tokeniser from BartTokenizer.
The tokeniser used was based on Byte-Pair-Encoding (BPE) [39], which creates a base vocabulary consisting of symbols occurring in the set of unique words in the training set. BPE learns how to merge symbols to form new ones, finds the most common symbol-pairs and adds these to the vocabulary to reduce the number of OOV words present. This is done until the target vocabulary size is reached.

Source and target sequences were tokenised in batches. They were padded to specified their specified `MAX_LEN` using special `<PAD>` tokens. This ensured that all encoded sequences were the same length, making it easier to calculate the overall token size and build batches. If sequences were longer than the maximum length, they were truncated.

The BartTokenizer splits a sequence into tokens available in the tokeniser vocabulary and converts these into IDs that can be understood by the model. The result of the tokeniser is a dictionary with keys `input_ids` and `attention_mask`. The attention mask is a binary tensor and shows the positions of any indices used for padding to ensure that the model does not attend to them as they provide no value. `input_ids` represent the token indices.

**Training**

I encountered some unforeseen issues due to the training set occasionally not having associated creator descriptions, as a result, the CustomDataset class handles this.
The training:test split is 80:20 and the split is conducted by taking a random sample of 0.8 of the whole dataset. The training set was also shuffled; this was useful as the nature of the dataset was to have episodes from the same podcast grouped together. Shuffled batches were more representative of the overall dataset. The model converged faster and the summaries generated had better ROUGE scores.
For each batch in the `training_loader` set, the loss function is obtained in the forward pass and computes $\frac{d \cdot loss}{dx}$, The optimisation algorithm I used was Adam [40], implemented by PyTorch, to optimise gradient descent and minimise the need for memory space. This was the default optimiser used as per Hugging Face. After training was completed, the validation function generated predictions based on the model parameters.

**Generation**

The generation methods employed were a Greedy Search algorithm and Beam search. Greedy search is a simple algorithm used for the decoding of seq-2-seq models, whereby we output a summary that has words with the highest probability at each position. By setting the `num_beams` parameter to 1, it defaulted to greedy search. Beam search is detailed in 3.7 and considered `early_stopping`. Setting this to `True` ensured that beam search is stopped when we have finished at least `num_beams` sentences per batch.

**Understanding the Repetition Penalty**

The repetition penalty represents how repeated tokens are penalised during the generation of summaries and is an implementation of the ideas introduced by Keshkar et al. [20]. Penalised

sampling considers the model distribution, uses a 'near-greedy sampling' and prevents repetitions through a penalty which discounts the scores of previously generated tokens. The number of words repeated from the preceding tokens is counted and weighted as follows:
An immediately preceding word had a count of 1, a word appearing two words behind the current one was weighted at 0.5, a word appearing three words behind the current one was weighted at 0.25 and so on.
This is a similar idea to coverage mechanisms which are detailed in section 3.7, the key difference being that repetition penalties are not applied during training.
A length penalty was also included: this encouraged the model to generate longer or shorter sequences depending on whether the penalty is $> 1$ or $< 1$. I encouraged generation of longer sentences to emulate the richness of human-generated summaries.

#### Decoding

The predictions are decoded via the `tokenizer.decode()` function where we skip special tokens and clean up the tokenisation spaces. Special tokens used by BART include:

- `bos_token`: beginning of sentence
- `eos_token`: end of sentence
- `sep_token`: used to separate different inputs
- `pad_token`: padding token

These predictions as well as their corresponding creator descriptions were written to a `.csv` to be evaluated via the ROUGE metrics.

## 3.4 Input filtering

Hugging Face Transformer models only allow inputs of certain lengths. BART models only consider the first 512 and for long transcripts that have a token size greater than 512, (86% of the transcripts), a lot of information is discarded. It is important to note that the number of tokens is not the same as the number of words.
Hence I tried a variety of methods for pre-filtering the data to pass in as input.

### 3.4.1 Custom Truncation

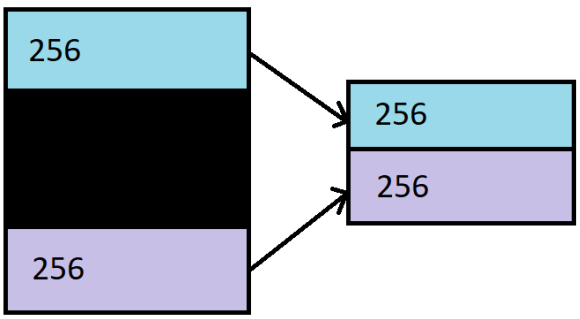


Figure 3.1: Image illustrating the truncation approach.

The motivation behind this approach was that I would be capturing information from the start and end of a transcript, leading to a broader range of informational content represented by the filtered transcript. I took 256 tokens from the start and 256 tokens from the end, as illustrated by 3.1. I applied the same tokeniser used by the BART model to first tokenise and then truncate the inputs. Unfinished sentences were discarded.
An example of an ASR-transcribed transcript truncated in this manner is 3.2.

```
Good night everybody and welcome to the first episode of play of bounds.
This is a new podcast that me and my friends Eddie and a co-star making
I'm Shaggy. I am your host for today and we're going to give you a
little overview of who we are. Thank you for tuning in for our first
episode of playoff bound and we hope to  hear from you guys soon. Thank
you. Yes, sir.  Appreciate it guys have a good night. Next episode of
the playoff bound podcast. All right. It's been a pleasure guys.
```

Figure 3.2: Truncated transcript example.

### 3.4.2 TextRank

TextRank[30] is an algorithm that is inspired by Google's PageRank that can be used as an extractive graph-based summarisation approach. The aim is to extract and rank the most 'relevant' sentences in a document. PageRank is used to calculate the weight for web pages. Web pages are connected to one another in a directed graph if they have links to each other. The formula that PageRank uses is:

$$S(V_i) = (1-d) + d \cdot \sum_{j \in In(v_i)} \frac{1}{|Out(V_j|} \cdot S(V_j)$$

Where $S(V_i)$ is the weight of webpage i, d is the damping factor, $In(V_i$ represents the set of inbound links of i and $Out(V_j$ represents the set of outbound links of j.
The key difference between PageRank and TextRank is what the nodes in the graph represent: in TextRank these are sentences. We include links between the sentences based on the similarities between the sentences. Similarity is measured as a function of the content overlap of two sentences. I considered the cosine similarities of the embeddings for all words within the sentences, as the measure of similarity.
The word embeddings used were GloVe embeddings [33]. GloVe (Global Vector for Word Representation) is an unsupervised learning algorithm for obtaining vector representations for words. I considered using word2vec but this disregards the word order in the sentences; important for maintaining accuracy and coherency of summaries. GloVe preserves global contexts as it creates a global co-occurrence matrix by estimating the probability a given word will co-occur with other words. Here for summarization the global context is a necessity. The GloVe embeddings[3] were pre-trained on aggregated global word co-occurrences from a Wikipedia corpus.
TextRank takes the weighted sum of all embeddings for the words in the sentence as the embedding for the sentence and builds the matrix based on the cosine similarity of these sentence embeddings.
The `networkx` library computes the PageRank algorithm on the graph generated from the similarity matrix using the following parameters:

[3]Obtained from nlp.stanford.edu/projects/glove/

- `alpha` = 0.85, the standard value used as the damping parameter.
- `max_iter` = 100, the maximum number of iterations.

I ranked the sentences by their scores and wrote them to a `.csv` file. The TextRank-ed sentences did not need to be truncated; the Hugging Face models handled truncation automatically. In this way the most relevant sentences of the transcript are passed into the model.

### 3.4.3 SpanBERT (*Extension*)

SpanBERT was first introduced by Mandar et al. [18]. It extends BERT's Masked-Language-Modelling by masking contiguous random spans of text rather than random tokens. Span boundary representations are also trained to predict the context of the whole span without considering tokens within the span. As per [18], it consistently outperformed BERT on span selection tasks such as question answering and coreference resolution. I hypothesised that as coreference resolution may enhance summarisation, SpanBERT summaries would be accurate with regards to its named entities.
I used the pre-trained `spanbert-base-cased` model and tokeniser, both from the Hugging Face transformers library. The cased model has the same benefits as that of the BERT model used. I ran SpanBERT with coreference resolution; this is the task of finding all expressions that refer to the same entity in a task. The model received embedded representations of each span in the document using SpanBERT. These span representations were scored and used to prune away spans that were unlikely to occur in a coreference cluster; the model decided which later spans the remaining ones are coreferent with.

```
handler = CoreferenceHandler(greedyness=.5)
spanbert_model = Summarizer(custom_model = model,
custom_tokenizer=tokenizer, sentence_handler=handler)
```

Coreference resolution is how we link together mentions that relate to real world entities.

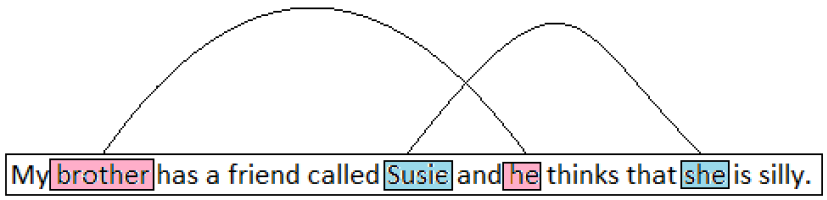


Figure 3.3: Illustration of coreference dependencies.

The `greedyness` parameter is a number between 0 to 1 which determines how greedy the model is when making coreference links. 0.5 is the default value.
The benefits of coreferences with summarisation can be seen clearly in the following example:

---

'Hi, I'm Jonathan I'll be discussing Mendy's performance. The Frenchman performed remarkably in the recent game and annihilated the competition'.

---

In this example we can see that the Frenchman and Mendy refer to the same person. Usually we have mentions that overlap with that of their antecedent and especially when they

are expressed with a proper name. Coreference resolution helps us to identify which entities are repeatedly mentioned, particularly useful for examples where they are expressed with different names, such as 'Mendy' and 'Frenchman'. Coreference is commonly used with information retrieval and retrieving referential relations between terms of the original query and the relevant terms of the documents.

## 3.5 T5 (*Extension*)

As an extension I decided to investigate the performance of Text-to-Text-Transfer-Transformer (T5) models and hence fine-tuned a T5 model on the dataset. The basis of this code is the same as that of the BART model- the only things that were changed were the tokeniser and the pretrained model. The pretrained model used is `t5-base`. The tokeniser used is `t5-base`, and is based on the SentencePiece tokenizer [21] which considers that non-space characters are used to separate words and then uses BPE to construct the vocabularies. The hyperparameters were kept the same as they were for fine-tuning BART.

## 3.6 LongFormer (*Extension*)

As an extension I considered a LongFormer model. Longformer models allow for longer input sequences than traditional Transformer models. This is because while traditional self-attention mechanisms scale quadratically with sequence length, Longformer mechanisms scale linearly. The Longformer self-attention has a local and global context such that most tokens attend locally to each other. This means that they attend to $\frac{w}{2}$ of their preceding and $\frac{w}{2}$ of their succeeding tokens where w is the specified window length. The local and global attention may be specified using a `global_attention_mask`.
I used a pre-trained Longformer model from
`patrickvonplaten/longformer2roberta-cnn_dailymail-fp16` and a pretrained tokenizer from `allenai/longformer-base-4096`. These were provided by the Hugging Face transformers library.

The encoder length is the maximum size of the input, 4096 tokens, as opposed to the 512 tokens that are considered with other Transformer models. I ran it with the Trainer class that is provided along with the Hugging Face library to enable faster training.
This involved creating the training dataset, tokenising the inputs as previously and then creating the `input_ids` and `attention_masks` for the individual batches. I then set 128 tokens to the global attention:

```
global_mask = [[1 if i < 128 else 0 for i in range(seq_len)] for seq_len
in len(source_ids) * [encoder_length]]
```

### 3.6.1 Loss function

The Longformer model was trained with ROUGE as its loss function. The other Transformer models that I considered language modeling loss, considering the number of `<MASK>` tokens that are predicted correctly. This led to summaries that scored better against ROUGE metrics as can be seen in 4.1.

## 3.7 Pointer Generator

The basis of this model was a seq-2-seq model; I made small modifications to an existing implementation given in

`https://github.com/ymfa/seq2seq-summarizer/blob/master/model.py`. The baseline model feeds the tokens into the encoder which is implemented as a single-layer bidirectional LSTM. The output of this is a sequence of encoder hidden states $h_i$. The decoder, which is implemented as a single layer unidirectional LSTM, takes the word embedding of the previous word of the reference summary and calculates the Bahdanau attention.
The context vector, which is a weighted sum of the encoder hidden states is calculated as follows:

$$h_t^* = \sum_i a_i^t h_i$$

The context vector is concatenated with the decoder state $s_t$ and is fed through linear layers to produce the vocabulary distribution as a means to generate a sequence word token by token:

$$P_{vocab} = softmax(V'(V[s_t, h_t^*] + b) + b')$$

### 3.7.1 Vocabulary

First I created a Vocabulary class to store all the unique words seen in the training set. Each word was given a unique index and stored in a dictionary, and GloVe embeddings were loaded for each word.
I also had to set aside specific tokens for reserved tokens, which are as follows:

- `<PAD>` tokens: Used to pad a token sequence to its maximum length specified. It is worth noting that token sequence length is different from the total number of words in a sequence as tokens may not be single words.
- `<SOS>` tokens: Used to indicate the start of a sentence.
- `<EOS>` tokens: Used to indicate the end of a sentence.
- `<UNK>` tokens: Unknown words. These tokens are ones that may appear in the test set and did not appear in the training sentence.

I created a OOVDict class to handle 'out of vocabulary' (OOV) words; words that are represented by the `<UNK>` tokens in the token representation. PG models are able to generate OOV words and hence there needed to be a way to store words that are not in the main vocabulary. The probability of an OOV word appearing in the vocabulary is zero and the attention is 0 as well.

### 3.7.2 Dataset

The dataset also had to be prepared. I handled filtering of the dataset in the same way that I handled this previously; removing entries where transcripts did not have corresponding creator descriptions. On receiving a file path as an argument, the instance of the Dataset is initialised and the following steps are taken:

- An instance of the Vocabulary class is initialised and the vocabulary is built up from the transcripts and episode descriptions as detailed in the Vocabulary section above.
- Batches of size 16 are built. It is worth noting that I intended to create batches of size 32, but the model continued to crash on Colab. The (transcript, creator description) pairs

were shuffled, the source and target tensors were initialised and were filled up by the indices of the words that they contain; this information is obtained from the Vocabulary class.

- An instance of OOVDict was also initialised and updated with `<UNK>`-labelled tokens.

### 3.7.3 Seq2Seq model

The Seq2Seq model consists of the following hyperparameters:

- `vocab`
- `max_dec_steps`: max num of decoding steps (only effective at test time, as during training the number of steps is determined by the `target_tensor`); it is safe to change `self.max_dec_steps` as the network architecture is independent of src/tgt seq lengths. This is set to 144
- `enc_attn`: this describes whether the decoder output depends on attention over encoder states. This is True, as we are applying attention
- `enc_attn_cover`: this describes whether provide the coverage vector as an input to the computation of attention over encoder states. This is True, as we are applying coverage
- `dec_attn`: this describes whether the decoder output depends on attention over past decoder states to avoid repetition. This is set to True
- `pointer`: this describes whether a pointer network is used or not. This is set to True
- `cover_loss`: coverage loss is multiplied by this value when added to total loss. This is set to 1
- `embedding`: this initialises the embedding layer
- `encoder`: this initialises the encoder layer
- `decoder`: this initialises the decoder layer

The model replaces any OOV index in the tensor created for the test set with an `<UNK>` token. The forward propagation involves partial forcing, such that for each step of the training we make a random decision as to whether to use teacher forcing or not.

**Teacher forcing**

Usually in sequence prediction, the output from the previous time step is used as the input for the model at the current time step. This recursive process is used often in model training but can lead to slow convergence and model instability. Teacher forcing works by using the real current target outputs from the training dataset as input at the next time step rather than the output generated by the network. Using teacher forcing causes it to converge faster but when the trained network is exploited, it may exhibit instability. The teacher forcing ratio is decreased by every batch, as per the forcing decay ratio:

$$\frac{k}{k + exp(\frac{i}{k})}$$

where k is the forcing delay and i is the batch number.
I also used partial forcing; this is the random choice between using teacher forcing or the model's own output; this happens at each training step rather than at each batch.

**Coverage**

I modified the existing model code to calculate coverage as per See et al. [38]. I combined the past attention weights into one single vector. This was as per the formula, taking the minimum of the coverage vector and the decoder attention.

$$covloss_t = \sum_i min(a_i^t, c_i^t)$$

The code was as follows:

```
cov_loss = sum(min(cov_vec, dec_enc_attn))
```

Coverage pertains to the amount of attention received so far and coverage loss penalises words that have been attended to. The mechanism has one part that is in the model architecture that considers the coverage vector to compute the attention and the other part is in the loss to prevent repeatedly attending to the same area of the input sequence. Coverage is calculated as follows:

$$c^t = \sum_{t'=0}^{t-1} a^{t'}$$

The code was as follows:

```
coverage_vector, _ = torch.sum(torch.cat(enc_attn_weights), dim=0)
```

Coverage considers the following hyperparameters:

- `enc_attn_cover`: describes to provide the coverage vector as an input to the computation of attention over encoder states
- `cover_func`: this describes which function (sum or max) should be used to aggregate previous attention distributions
- `cover_loss`: we multiply the loss by this value when we add it to the total loss; it works like a weighting

I calculated the attention and vocabulary distribution as well as the generation probability which is the probability of generating a word from the vocabulary vs copying a word directly from the source text.

```
output = (prob_gen * gen_output) + (1-prob_gen) * (enc_attn)
```

So the probability of producing a word is the probability of generating it from the vocabulary · the generation probability + the probability of pointing to it anywhere it appears in the source text · the copying probability.

**Decoding**

Code was also written to handle the decoding and evaluation of the generated summaries. The decoding took place by using the Vocab instance in order to translate token indices from the summary hypothesis generated by the model and selected by beam search into words. Evaluation 3.4 of the summaries was based on ROUGE metrics that are detailed in the Preparation chapter 2.3.1 and later this chapter 3.8.1.

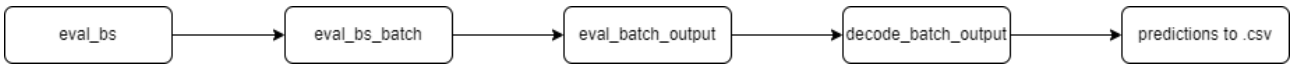


Figure 3.4: Evaluation pipeline.

### 3.7.4 Beam Search

Beam Search is a pruned down version of a traditional breadth-first search, parameterised by search width k. It takes the starting word, considers all possible following words and stores the k 'best' or most likely words. This is done by using the encoder's output to generate the probabilities for each position. At position k, the model is rerun k times to generate probabilities by fixing each possible preceding token sequence, to produce k branches with k sets of probabilities. This is done until we reach an `EOS` token as the best character for a position. The 'best' sequence has the highest probability and is output.
Greedy Search is Beam Search with k=1. It is trivial to see why this fares badly; word probabilities may not be accurate initially and so the first words generated may be incorrect. These errors will propagate through each step.
I encoded inputs into hypotheses, ran the decoder over batches and created new batches based on the decoders' results. Hypotheses that were longer than the minimum length and shorter than the maximum length were returned.

### 3.7.5 Hypothesis

The purpose of the Hypothesis class is to store all possible summaries for a given transcript. The summary with the highest probability or score is chosen during beam search. Each instance of Hypothesis contains:

- `tokens`: this is an ordered list of tokens that pertain to the summary that has been generated.
- `log_probs`: log probabilities for each token generated in the summary.
- `dec_hidden`: the decoder hidden states.
- `dec_states`: the decoder states.
- `attn_dists`: the weights for the attention mechanism in the encoder.
- `coverage`: the coverage calculated at a point.

### 3.7.6 Training

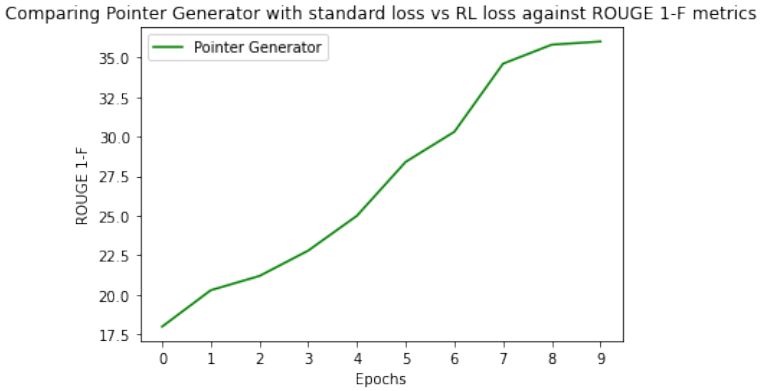


Figure 3.5: Performance of PG over 20 epochs.

I trained the model over a period of two weeks, for 20 epochs. PG models had very poor performance when trained over 4 epochs, hence I trained the model for more epochs than the other models. The ROUGE scores of the resulting summaries improved steadily for each epoch trained, as shown in 3.5.

The optimizer used in training the model was the Adagrad optimiser implemented within the PyTorch library. Training occurs in batches. For each batch, the NLLLoss (negative log likelihood loss) is computed.

### 3.7.7 Reinforcement Learning (RL) Loss (*Extension*)

As an extension I implemented an RL loss function. If the `rl_ratio` variable was set to $> 0$, I randomly sampled the output and obtained the ROUGE score for evaluating the output. I subtracted the actual ROUGE score for the target from the ROUGE score for the summary generated by the model; forming the basis for the loss function. If the sample $>$ the baseline then the reward is positive leading to good exploration and hence the `rl_loss` is negative. The standard loss was combined with the RL loss in proportion to the `rl_ratio` that was specified.

```
rl_loss = neg_reward * samp.loss
rl_loss_value = neg_reward * samp.loss_value
loss = (1 - rl_ratio) * out.loss + rl_ratio * rl_loss
loss_value = (1 - rl_ratio) * out.loss_value + rl_ratio * rl_loss_value
```

## 3.8 Code for Evaluation

### 3.8.1 ROUGE

This was implemented by using the rouge library and getting the average ROUGE-1, ROUGE-2 and ROUGE-F scores for each set of summaries that were generated.

### 3.8.2 Perplexity

I used the nltk library to compose the training data, which in this case, was a sample of the COCA database, into bigrams. I considered a bigram model as we cannot consider words in English as occurring independently hence considering jointly conditional probabilities was more

appropriate for considering a corpus of natural language.
I built the vocabulary from all the words in the training set and then initialised and fitted a Laplace model to the data. Using the creator description data as the test data, the model outputs the average perplexity score for the summaries. The perplexity calculation is as described in 2.3.2.

## 3.9 Repository overview

This project involves 5 main modules: `bart`, `preprocessing`, `evaluation`, `pointer_generator`, `misc` that feed into each other as explained in B. `preprocessing` is described in 3.1, `bart` is described in BART 3.3 and T53.5 and comprises code for training BART and T5 models, `misc` comprises code for training the BERT model with k-means clustering for the embeddings 3.2, the Longformer model 3.6 and the SpanBERT model 3.4.3. `pointer_generator` consists of code for implementing the PG model, described in 3.7 and `evaluation` contains code for perplexity and ROUGE metrics.

```
spotify_podcast_summary_challenge/
├── data/
├── bart/
│   ├── bart.ipynb
│   ├── custom_truncation.py
│   └── textrank.ipynb
├── evaluation/
│   ├── perplexity_meteor.py
│   └── rouge_evaluation.py
├── misc/
│   ├── bert_k_means.ipynb
│   ├── longformer.ipynb
│   └── spanbert.ipynb
├── pointer_generator/
│   ├── pg_model.ipynb
│   └── model.ipynb (modified from github.com/ymfa/seq2seq-summarizer
│   /blob/master/model.py)
└── preprocessing/
    ├── bart_train_regex.py
    ├── extract_and_filter _regex.py
    └── svm.ipynb
```

# Chapter 4

# Evaluation

## 4.1 Success Criteria

All of the success criteria set out for this project were met and are detailed in the Appendix A. I have summarised the core success criteria below:

- Preprocessing the data before training and removing transcripts with ads[1].
- Fine-tuning the BART model[1].
- Implementing the Pointer Generator model[1].
- Evaluating the models using ROUGE metrics.

This chapter details the fourth criterion.

## 4.2 Overview

The evaluation of summaries can be conducted in qualitative and quantitative ways. In my project I decided that it was important to consider both of these approaches as they both measure different things. Qualitative evaluation can give us a sense of the linguistic quality or readability of the generated summary while quantitative evaluation measures the informativeness or the content coverage of the summaries.
I provide a holistic evaluation of the summaries; ROUGE scores expressing the degree of similarity between generated and reference summaries, perplexity scores expressing the readability of the summaries and human-evaluation to rate coherency and accuracy.

### 4.2.1 Interpreting the results

ROUGE scores: F1 scores are considered here, meaning that the scores reported express the degree of similarity between the model-generated summaries and reference summaries/creator descriptions. I considered the test set provided in the dataset, consisting of 1027 transcripts and reference summaries. The closer to 100 the ROUGE scores are, the higher the summarisation quality.
Perplexity: Perplexity can be interpreted as the weighted branching factor. The higher the perplexity score, the more confused the model is when trying to guess the next word. A low perplexity score is considered a measure of good readability.

[1]Described in 3.

## 4.3 Results

### 4.3.1 Quantitative Results: ROUGE and Perplexity

| Model | ROUGE 1-F | ROUGE 2-F | ROUGE L-F | Perplexity |
|---|---|---|---|---|
| **Fine-tuned BART (standard truncation)** | 28.3 | 13.2 | 25.9 | 13.4 |
| **Fine-tuned BART (TextRank)** | 31.0 | 14.5 | 29.3 | 13.6 |
| **Fine-tuned BART (custom truncation)** | 18.2 | 3.5 | 17.8 | 13.5 |
| **Fine-tuned T5 (standard truncation)** | 25.5 | 14.2 | 25.4 | 14.2 |
| *T5, Zheng et al [45]*[1] | *22.2* | *4.5* | *17.5* | - |
| **Fine-tuned T5 (TextRank)** | 33.0 | 13.0 | 30.1 | 14.1 |
| **Fine-tuned T5 (custom truncation)** | 17.6 | 3.3 | 17.3 | 14.3 |
| **Finetuned Longformer** | 20.3 | 10.1 | 18.9 | 13.2 |
| **Fine-tuned T5 with SpanBERT** | 24.0 | 14.3 | 23.0 | 57.2 |
| **BERT with k-means** | 10.6 | 5.1 | 8.1 | 65.0 |
| **PG** | **36.0** | 14.2 | 33.0 | 23.4 |
| *PG, See et al [38]*[1] | *39.5* | *17.3* | *36.4* | - |
| **PG with RL** | **38.4** | **15.6** | **35.5** | **89.3** |
| **TextRank** | 12.2 | 2.1 | 9.9 | 45.6 |

Table 4.1: Average **ROUGE-1F**, **ROUGE-2F**, **ROUGE-LF** and **perplexity** scores for generated summaries.

One thing to note is that the ROUGE-F1 scores are heavily influenced by the length of the output summaries, as we divide the number of matching n-grams by the length of the summary. Hence, shorter summaries are more likely to have higher ROUGE scores. The PG model generated summaries that tended to be shorter than the others, this could be down to having to specify a minimum summary length for Beam Search. If there were no summaries within the desired length parameters, Beam Search would return any truncated summaries. The other models did not require this minimum length parameter.

**PG:**

As we can see from 4.1, PG with RL loss generated summaries that maximised all three ROUGE metrics. If other models had also been trained for 20 epochs this may not have been the case. These summaries had the highest perplexity score: 89.3. There are two possible implications here. Firstly, that these were the least readable, and secondly, that other models stuck to predictable generation whereas PG with RL was more diverse in its use of phrases. High perplexity could be indicative of varied language, thus making them more interesting and human-like.

From 4.1 we observe how the PG with RL loss consistently outperformed the standard loss

[1]These papers did not include scores for perplexity in their results.

function for all epochs. It is worth noting that this trend is only observable for the first 20 epochs; had I had more time on the project I would have trained the models for more epochs to try and match or overtake the See et al. ROUGE scores.

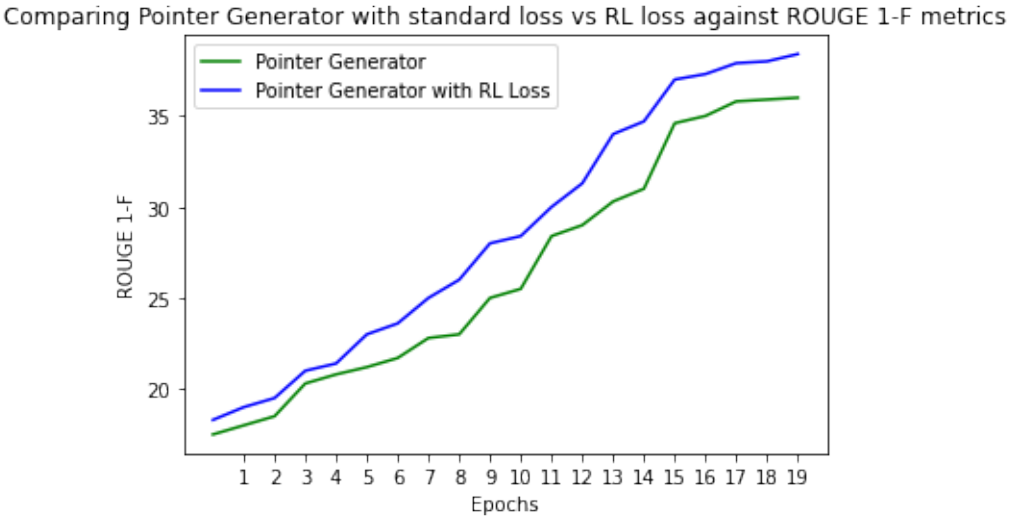


Figure 4.1: Performance of PG vs PG with RL loss.

**Input filtering methods**

The best filtering method was TextRank, 4.1 shows that the perplexity scores for the BART and T5 models were low (respectively 13.6 and 14.1) and the ROUGE scores were maximised for all metrics in comparison to the standard and custom truncation methods. The BART summaries and T5 summaries respectively had similiar perplexity scores, indicating that the model and training method had more weight on the readability rather than the actual input.

**Longformer**

The Longformer model did not eclipse the performance of the T5 and BART models; surprising given that there should have been more overlap of n-grams given the larger span of content considered as input.

**BERT**

Summaries generated with BERT with k-means scored poorly. I speculate that as this model clustered similar sentences together, the truncation that took place when the sentences were passed in as input meant that there was a lot of overlapping content. This might have led to less new informational content. Hence there is less overlap with the creator descriptions, corresponding to the lower ROUGE scores. BERT was used as an extractive model. The high perplexity scores might reflect that the transcripts themselves were not readable.

**SpanBERT**

SpanBERT's poor performance is surprising as it is supposed to output spans of text that are representative of the different ideas presented in the input. It might have been more difficult to find relevant spans of text due to the colloquialisms and lack of coherency in the input.

**Comparison to other models**

Comparing the ROUGE scores of PG with those from See et al., there was only a difference of (3.5 |3.1 |3.4) in terms of the metrics. I managed to get results within 10% of See et al's scores

for ROUGE 1-F and ROUGE L-F.
I significantly improved the model from Zheng et al. [45] by (49%|190%|72%) for the respective ROUGE metrics4.1. These improvements have been obtained by using TextRank for the input. Zheng et al. considered 60% more podcasts leading to potential for greater variation which may explain why my results eclipsed those of the paper.

### 4.3.2 Quantitative Results: METEOR

I then decided to take on further evaluation to inspect the differences between certain summaries more closely. I considered METEOR as a metric to use between sentences generated by T5 and BART, as well as PG and PG with RL. METEOR (Metric for Evaluation of Translation with Explicit ORdering) [22] is traditionally used for the evaluation of machine translation output. The algorithm creates alignments (mappings between ngrams) between two sentences and chooses the one with the fewest intersections of mappings. METEOR consistently demonstrates high correlation with human judgments in independent evaluations.

| **Comparisons** | **Average METEOR score** |
|---|---|
| **BART and T5** | 0.23 |
| **PG and PG with RL** | 0.31 |

Table 4.2: Average **METEOR** scores for different models.

I am surprised at the lack of correlation between summaries generated by PG and PG with RL; this illustrates the effect that different loss functions have. While BART and T5 score similarly against all other metrics, they have low relatedness. This indicates that there are clearly a number of ways to create similiar-scoring summaries.

### 4.3.3 Greedy vs Beam Search

When training the BART model, I experimented with Greedy Search and Beam Search to decode and generate the summaries. I eventually settled on Beam Search as the final method but below are my findings when considering their performance with respect to ROUGE metrics:

As we can see from , Beam Search consistently outperformed Greedy Search when it came to the ROUGE metrics

### 4.3.4 Qualitative results

Human evaluation is good way to assess readability, coherency, grammaticality and the focus and structure of the given summary. Such manual assessment is very laborious and hence I enlisted the assistance of 10 human volunteers[1] who answered a mix of open and closed questions. The questions that were asked are detailed in Appendix C.
Each participant was given 2 'summary bundles' to evaluate. Each bundle contained the transcript, creator description and a selection of the generated summaries. The ones that I considered were as follows: Fine-tuned BART with standard truncation, Fine-tuned BART with TextRank, Fine-tuned T5 with standard truncation, Fine-tuned T5 with TextRank, Fine-tuned T5 with SpanBERT, Fine-tuned Longformer, BERT with k-means clustering, PG(not with RL) and TextRank.

[1]I obtained explicit ethics approval from the CST committee in advance of carrying out these user studies.

| | BART with standard truncation | BART with TextRank | T5 with standard truncation | T5 with TextRank | T5 with SpanBERT | Longformer | BERT with k-means | Pointer Generator | Text Rank |
|---|---|---|---|---|---|---|---|---|---|
| **Contained Named Entities** | 20 | 20 | 20 | 20 | 13 | 20 | 7 | 20 | 13 |
| **Contained Inaccuracies** | 4 | 1 | 3 | 1 | 11 | 2 | 11 | 0 | 0 |
| **Contained Key Facts** | 17 | 19 | 17 | 20 | 12 | 20 | 8 | 20 | 6 |
| **Easy Genre Discernment** | 17 | 18 | 18 | 18 | 5 | 17 | 3 | 20 | 4 |
| **Contained Wording Peculiarities** | 0 | 0 | 0 | 0 | 14 | 0 | 5 | 0 | 7 |
| **Was Coherent** | 19 | 19 | 20 | 20 | 18 | 18 | 12 | 20 | 12 |

Table 4.3: Summary of the results of the human evaluation, scores are out of 20.

**Fine-tuned BART Fine-tuned T5 with standard truncation**:
Both of these models performed well with regards to capturing the tone of the episode, and the key facts (19/20) and (20/20) respectively as per 4.3. Key facts tend to crop up near the start of the episode during the introduction but for the more rambly podcasts this was not quite the case. The English was coherent (19/20) and (20/20) as per 4.3 and overall the summaries were very succinct and did not include unnecessary details to confuse the readers.

**Fine-tuned BART Fine-tuned T5 with standard truncation**:
These summaries had all the positives of the standard truncation BART and T5 models along with capturing the key facts better for more rambling episodes that lacked sharp focus, (17/20) as per 4.3. It is clear that TextRank was the most effective filtering method that I used.

**Custom truncation**[1]:
Of the small sample inspected, I noticed that coherency was low, and the sentences generated were less intuitive. This was surprising given the low perplexity scores, indicating that perplexity measures were not necessarily synonymous for excellent readability. This highlights the importance of human judgement.

[1]I evaluated 5 bundles myself due to poor ROUGE scores obtained and not wishing to increase the scope of the human study.

**T5 with SpanBERT**:
The summaries were generally of mixed quality. 14/20 of the inspected summaries 4.3 contained wording peculiarities, lots of incorrect information and infrequent discernment of the key facts. These summaries were coherent.

**Longformer**[2]:
These summaries were of a high quality; as a much longer segment of the transcript was considered during training they grasped the key features of the episodes 4.3. Some episodes were very conversational, lacked focus and explored a range of topics; these were summarised well by the Longformer.
Grammar, coherency and readability were of good quality as well, although there was some repetition.
The evaluation of the Longformer models is contradictory, the ROUGE metrics indicated that these summaries were less similar to the creator description than the BART and T5 models, but the inspection deemed these to be the best model. This highlights the fallibility of an important assumption made earlier: that creator descriptions are the gold standard.
Clearly, at times, the summaries may be of a higher quality than the ones given.

**BERT with k-means clustering**:
The summaries tended to perform poorly when it came to capturing the key features of the transcript; only 8/20 did this 4.3. Some sentences that BERT chose failed to actually make sense, eg)

```
"You cannot scummy hashtag wisely, or the people's Mentor in today."
```

The summaries were very poor, as reported almost unanimously by the participants, and very colloquial in nature, frequently starting informally like 'Hey it's Nick!'. Coherence was also poor at times, and the summaries suffered from repetition.

**PG**:
The summaries tended to have the best results, all were deemed coherent 4.3, all contained key features of the source transcripts 4.3 and captured relevant named entities better than others. This may be due to the way that I implemented the model to consider OOV words with the pointer mechanism. There tended to be very little repetition as well.

**PG with RL**[2]:
Participants reported that the summaries contained a lot of the key content but were not very readable, corroborating the perplexity scores. The ends of sentences were not coherent and the most common issue was the presence of short and truncated sentences towards the ends of sequences. I hypothesise that this was due to using a single discrete evaluation metric (ROUGE-1) as the loss function.

**TextRank**:
As we can see from 4.3, TextRank rarely contained named entities, and tended to be very affected by incoherent and colloquial text. Incoherencies in the source text arose as a result of the transcriptions being directly transcribed by ASR, conflicting voices and interruptions rendering some fragments less readable than others. 8/20 summaries were deemed incoherent 4.3. As this approach captures the most 'relevant' sentences as per the algorithm, there was also a lack of punctuation and poor sentence termination, as exhibited by the following fragment:

[2]I evaluated 5 bundles myself due to time constraints.

```
"Let's download these stories about whoever I don't care Company products
could be don't like people you hate hot just getting people you do like
whatever Compensation Plan stories both stories how stories car stories
truck stories debt stories electricity storage"
```

TextRank also included instances of repetition, for example

```
'But you can touch his life. But you can touch his life'
```

as this was a sentence repeated for emphasis in different locations in the source text. Participants mentioned that the summaries felt 'abrupt' and read like random snippets of text where genre discernment was poor.

### 4.3.5 Named entities

Named entities are real-world objects, such as people, locations and organisations that tends to be denoted with a proper noun. The more named entities that are preserved in a summary, the more 'specific' the summary may be described as being. For example, the summary beginning: 'In today's episode they speak about football team' is less descriptive than 'In today's episode Fran and Mark speak about the Arsenal football team'.

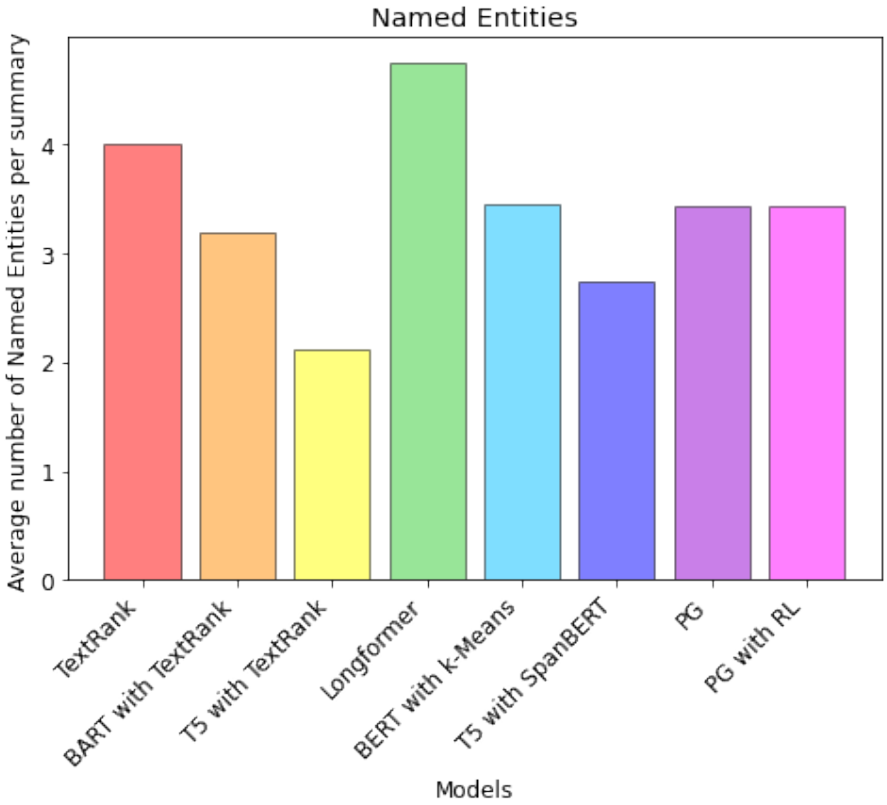


Figure 4.3: Average number of named entities per summary per model.

I also decided to investigate which models preserved the most distinct named entities; the results are surprising. Longformer scored the highest as can be seen as per 4.3. I hypothesise that this is as a result of more of the episode transcript being considered and hence more named

entities being introduced over the span of text. T5 with TextRank scored the lowest on average, a surprising result given the perception of the human volunteers in comparison to BART.

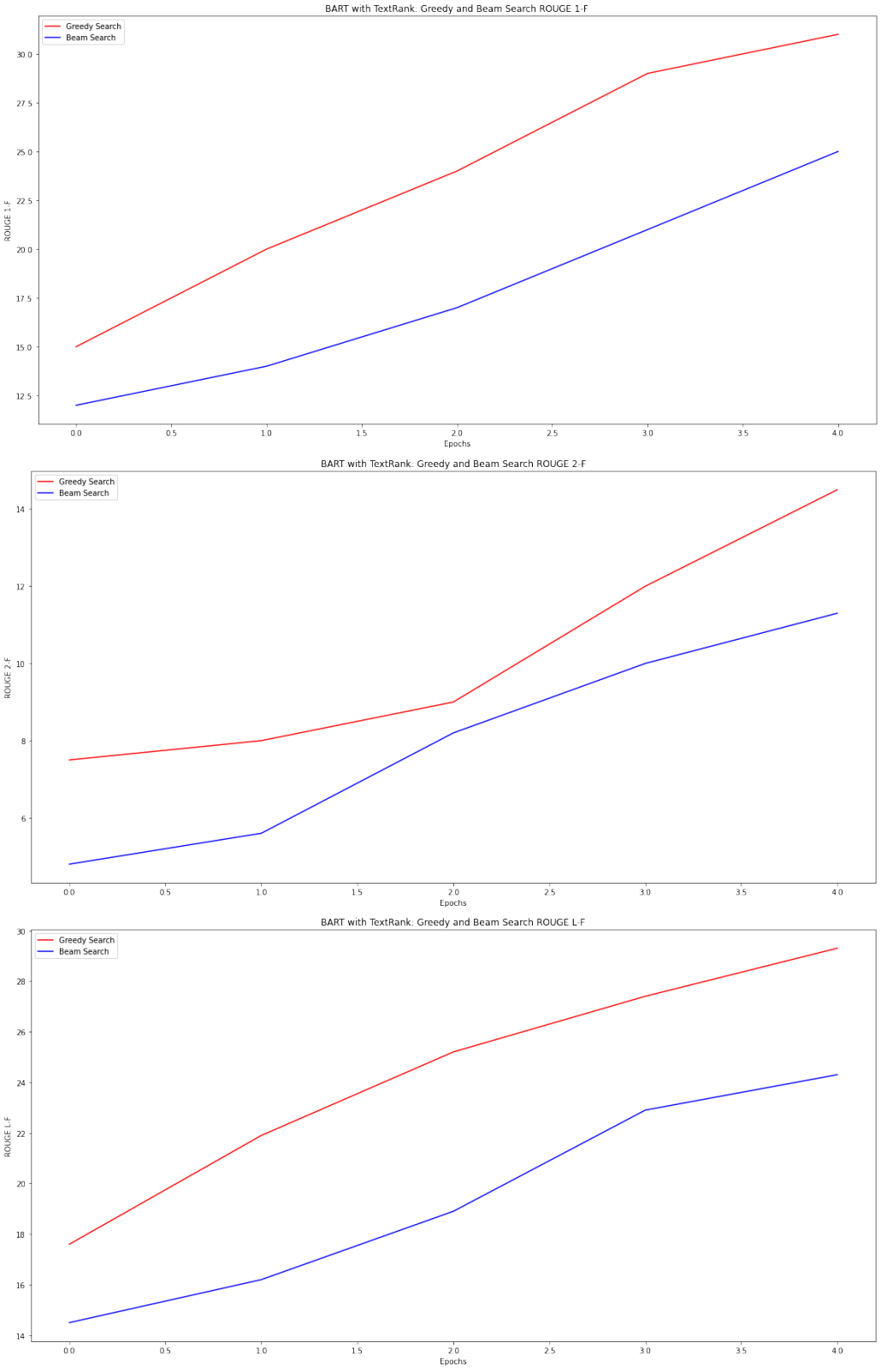


Figure 4.2: ROUGE performance of Greedy and Beam Search over 4 epochs.

# Chapter 5

# Conclusion

## 5.1 Achievements

In this project I have completed the summarisation task from the Spotify Podcast Dataset Challenge and in doing so created an array of summarisation models that have been evaluated against ROUGE metrics and user studies. I successfully preprocessed the Spotify dataset using a SVM and fine-tuned and implemented abstractive models (respectively BART and Pointer-Generator) and experimented with text filtering methods, such as TextRank and SpanBERT. In completing these tasks I successfully met all of my success criteria. I completed extensions; fine-tuned further abstractive models (T5, Longformer) and extractive models (BERT), considered RL loss functions and considered METEOR and perplexity evaluation metrics.

The model with 'best' performance was PG; it gave ROUGE scores of (36.0|14.2|33.0) and was rated highly by the volunteers on a number of aspects. The best filtering method proved to be TextRank, compared to standard and custom truncations (defined in 3.4.1). My T5 model had a (49%|190%|72%) improvement over Zheng et al. against ROUGE-1F, ROUGE-2F, ROUGE-LF metrics [38]. My PG model was within 10% of the ROUGE-1F and ROUGE-LF scores obtained by See et al.

## 5.2 Personal Reflections and Lessons Learnt

Over the course of the project, I familiarised myself with many new Python libraries and amassed a lot of knowledge in the fields of Summarisation and various architectures such as BERT, BART and Pointer-Generator models.
I also learnt to appreciate the computational resources needed to train these models. If I were to have my time again with the project, I would investigate further approaches to optimise the training process. This would have enabled even more experiments.
As this was the biggest project that I have undertaken thus far, I was pushed outside my comfort zone and learnt how to quickly skim and discern the contents of academic papers to create my own implementations. I feel empowered that I was able to carry out such a substantial project and manage my time well.

## 5.3 Future Work

If I had more time I would like to train all models with TextRank-ed and SpanBERT-ed inputs; TextRank performed very well and I did not manage to test the effects of SpanBERT on models other than T5. In the interests of time and computational resources I had to curtail the training

of the models to fewer epochs than I would have preferred. The performance of PG steadily increased over epochs so training over more epochs could have yielded even better results.
I hypothesised in 4.3 that the lack of readability of summaries generated by Pointer-Generator with RL was as a result of optimising for only one metric. With more time I could augment the loss function to also take perplexity into account. Li et al. [24] considered a similar type of RL loss function, combining more than one metric.
Another exciting aspect to consider in the future may be multimodal analysis. The dataset included audio files information on prosody which may have been an interesting way to filter out the input to the various models to enrich information that the model learned.

The findings of this project may be used as a basis for further research into the interesting field of summarisation with informal corpora.

# Bibliography


[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer, 2020.

[4] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[5] Asli Celikyilmaz, Antoine Bosselut, Xiaodong He, and Yejin Choi. Deep communicating agents for abstractive summarization. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1662–1675, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[6] Ann Clifton, Sravana Reddy, Yongze Yu, Aasish Pappu, Rezvaneh Shadi Rezapour, Hamed Bonab, Maria Eskevich, Gareth Jones, Jussi Karlgren, Ben Carterette, and Rosie Jones. 100,000 Podcasts: A Spoken English Document Corpus. pages 5903–5917, 01 2020.

[7] C. Vapnik Cortes. Support-vector networks, 1995.

[8] Mark Davies. Corpus of Contemporary American English (COCA), 2015.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

[10] Published by Spotify Engineering. Introducing the Spotify Podcast Dataset and TREC Challenge 2020, Jan 2022.

[11] Frederic B. Fitch. Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *Journal of Symbolic Logic*, 9(2):49–50, 1944.

[12] Maria Fuentes Fort, Enrique Alfonseca, and Horacio Rodríguez. Support Vector Machines for Query-focused Summarization trained and evaluated on Pyramid data. 01 2007.

[13] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[14] Yue Guo, Wei Qiu, Yizhong Wang, and Trevor Cohen. Automated Lay Language Summarization of Biomedical Scientific Reviews, 2020.

[15] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching Machines to Read and Comprehend. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[16] Tsutomu Hirao, Hideki Isozaki, Eisaku Maeda, and Yuji Matsumoto. Extracting important sentences with support vector machines. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[18] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020.

[19] Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition.* Pearson Prentice Hall, Upper Saddle River, N.J., 2009.

[20] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

[21] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[22] Alon Lavie and Michael J. Denkowski. The Meteor Metric for Automatic Evaluation of Machine Translation. *Machine Translation*, 23(2–3):105–115, sep 2009.

[23] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension, 2019.

[24] Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. Deep Reinforcement Learning for Dialogue Generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202, Austin, Texas, November 2016. Association for Computational Linguistics.

[25] Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[26] Yang Liu. Fine-tune BERT for Extractive Summarization, 2019.

[27] J.N. Madhuri and R. Ganesh Kumar. Extractive Text Summarization Using Sentence Ranking. In *2019 International Conference on Data Science and Communication (IconDSC)*, pages 1–3, 2019.

[28] Potsawee Manakul and Mark Gales. CUED_speech at TREC 2020 Podcast Summarisation Track, 2020.

[29] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[30] Rada Mihalcea and Paul Tarau. TextRank: Bringing Order into Texts, Department of Computer Science University of North Texas. *Archived from the original*, 2004.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[32] Romain Paulus, Caiming Xiong, and Richard Socher. A Deep Reinforced Model for Abstractive Summarization, 2017.

[33] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[34] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer learning with a Unified Text-to-Text Transformer, 2019.

[35] Rezvaneh Rezapour, Sravana Reddy, Ann Clifton, and Rosie Jones. Spotify at TREC 2020: Genre-Aware Abstractive Podcast Summarization, 2021.

[36] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[37] Mike Schuster and Kaisuke Nakajima. Japanese and Korean Voice Search. In *International Conference on Acoustics, Speech and Signal Processing*, pages 5149–5152, 2012.

[38] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

[39] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

[40] KAREN SPARCK JONES. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[41] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[43] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[44] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.

[45] Chujie Zheng, Harry Jiannan Wang, Kunpeng Zhang, and Ling Fan. A baseline analysis for podcast abstractive summarization. *arXiv preprint arXiv:2008.10648*, 2020.

# Appendix A

# Success Criteria

The core components of the system can be summarised as follows:

- Handling the preprocessing of the data in order to prepare for the finetuning and training of the models that I implemented. I aim to implement a Support Vector Machine (SVM) in order to do this.
- Fine-tuning a pre-trained Bidirectional and Auto-Regressive Transformer (BART). [23] model. This involves passing the model a selection of transcripts and summaries from the Spotify Podcast dataset to attune it to the domain. BART is a common state-of-the-art approach to the abstractive summarisation tasks. I will experiment with various filtering methods such as TextRank and different modes of truncation to provide input into the model.
- Implementation and training of a Pointer Generator model, akin to the one implemented by See et al (2017).
- Using Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [25] metrics, which is the standard evaluation method for summarisation. I will also carry out qualitative evaluation using human volunteers.

# Appendix B

# High Level Visualisation of Project Repository

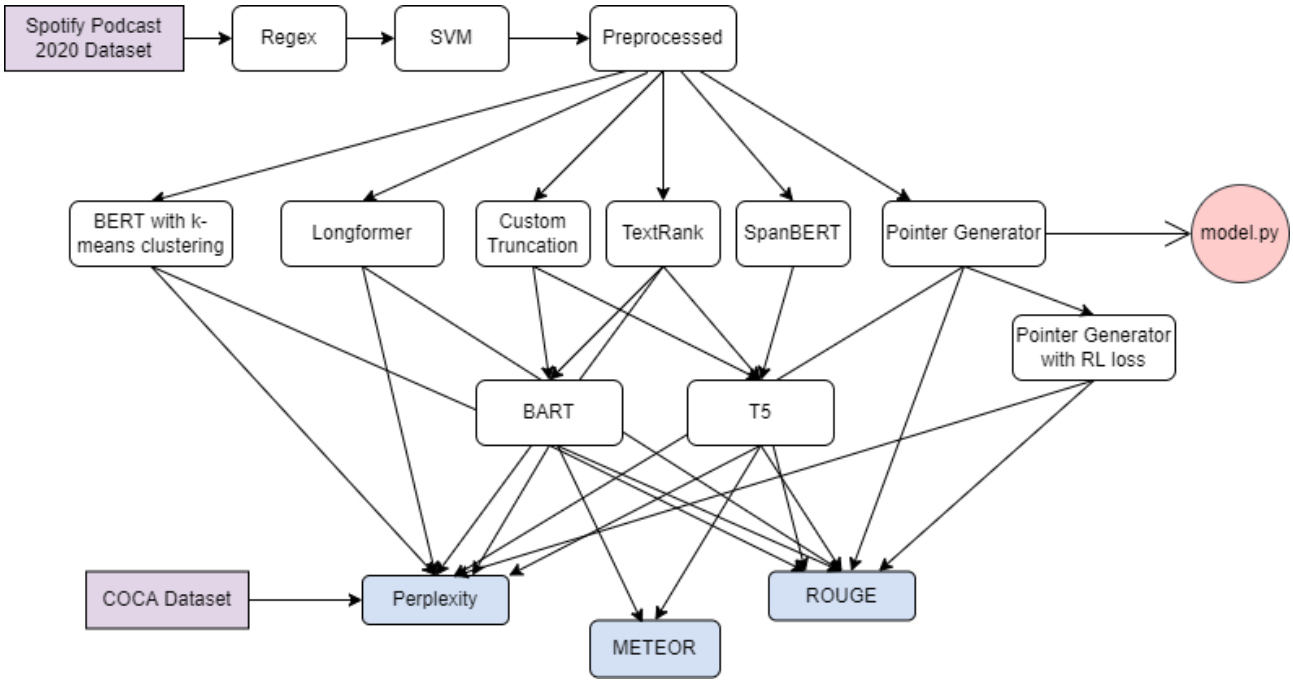


Figure B.1: A high level overview of my project, purple boxes are datasets, blue boxes are modules corresponding to evaluation, pink circles correspond to code that I did not write.

# Appendix C

# Human evaluation: Questions asked

1. Does the summary include named entities ie. names of hosts?
2. Does the summary include any incorrect data? Please list any inconsistencies with references to the original transcript or the summary? For example: the transcript/description reports that 'the Bulls lost 2-0' as opposed to the summary saying that 'the Bulls won 2-0'.
3. Does the summary contain all the key facts, in your opinion? What are some key facts you would have included in the summaries?
4. What information, if any, does the summary leave out? Do you think this provides for some suspense?
5. How likely are you to agree with the statement: 'The summary captures the tone and genre of the podcast, ie. suspense for true crime etc.' on a scale of 1-5, 1 being highly disagree, 2 being disagree, 3 being neither agree nor disagree, 4 being agree and 5 being highly agree.
6. Does the summary include the main topic of the podcast, ie. can you discern which genre the podcast pertains to.
7. Are there any wording peculiarities? Or oddities that would not be present in a standard summary, for example, the summary beginning with 'Welcome listeners!'
8. Is the podcast in coherent English? Do all sentences start and terminate in an appropriate way? Please list any examples of poor English.
9. Of the summaries generated, which is the 'best', and why. Describe what you think makes a summary good.

# Appendix D

# Sample Summaries Generated

**TextRank**

| |
|---|
| Yeah because I know I get really really pissed off free really easy that's being like it's all about like, you know, it's sometimes people don't get really get angry for one reason because I say, I actually get angry because I bought up a lot of stuff then like when I get angry at so So magical smacking so like, you know, sometimes people think when you are angry like you adjust your bullshit toy, but it's not like that angers part of like this emotion you have to have in your life. I have to say this on the podcast like to me I think all the stuff and going through it's not just unique people to feel like yeah, you can you can do it like just but it's just to tell you like I'm going through it and he can't even watch it go through and it's fine if you're not feeling okay, or you're not feeling all right, for the meantime, you know, you're helping people in the wake of the modern art podcast is having a knock-on effect and it's beneficial in good effect. |

**BART with TextRank**

| |
|---|
| This week we talk about anger and how to deal with it. In this episode, I talk about anger and how to deal with it. |

**T5 with TextRank**

| |
|---|
| In this episode, I talk about anger and how to deal with it. |

**T5 with SpanBERT**

| |
|---|
| This is a great episode and we hope you enjoy it. We love you guys so much for listening to our podcast! If you like what we do, check out our Patreon. |

**Longformer**

| |
|---|
| In this episode, I talk about how to deal with anger and how it can be harvested for greater good. Rapper De La Soul talks about how he gets angry and to finish the Beast. |

**BERT with k-means**

Did you always call me des and I have developed for the second time today. Yeah because I know I get really really pissed off free really easy that's being like it's all about like, you know, it's sometimes people don't get really get angry for one reason because I say, I actually get angry because I bought up a lot of stuff then like when I get angry at so So magical smacking so like, you know, sometimes people think when you are angry like you adjust your bullshit toy, but it's not like that angers part of like this emotion you have to have in your life.

**Pointer Generator**

We talk about anger, how it is harvested and how we deal with it. In this week's episode, rapper De La Soul talks about anger. He explains why he gets angry and how he deals with it. In the end, De La Soul says he wants to finish the Beast.

# Appendix E

# Project Proposal

# Automatic Generation of Podcast Summaries from Episode Transcriptions: Spotify Podcast Dataset Challenge

## E.1 Introduction and Description of the Work

In recent years, podcasts have risen in popularity as a media source and Spotify is one of the current leaders in the podcast industry. As there are so many podcasts out there, it is often hard to find particular episodes that are tailored to one's interests and hence the need for effective automatic summarisation as a tool for analysis is growing.

Podcast summaries greatly inform which episodes a listener chooses to listen to, and human authored podcast descriptions do not always provide useful summaries, varying greatly in quality from podcast to podcast. My chosen project involves producing concise summaries based on episode transcriptions. The summaries should include the key contents of an episode, as per the rules of the Spotify Podcast Dataset Challenge. [1]

This is an interesting and challenging task because NLP summarisation has largely focussed on news articles containing formal language; podcasts tend to be dialogues with many colloquialisms and noisy with sponsorships and advertisements. Genre specific abstracts vary in style as well; sports podcast summaries are more factual and include names and in depth whereas true crime focussed podcasts tend to leave out information so as to provide an air of mystery and intrigue. I will need to ensure that the nature of the podcast summaries fit the genre of the episode.

Beyond the immediate focus, which is to produce a solution for the Spotify Podcast Dataset Summarisation Challenge, the work done here could be used as the basis for other

[1] https://engineering.atspotify.com/2020/04/16/introducing-the-spotify-podcast-dataset-and-trec-challenge-2020/

dialogue summarisation. I plan on investigating extractive and abstractive summarisation methods and evaluating various models against each other.

## E.2 Starting Point

- Python: I have experience coding in Python and have used it for various personal projects as well as in Part 1A in Scientific Computing and in Part 1B in Foundations of Data Science.
- PyTorch, TensorFlow: I have used PyTorch and TensorFlow before for personal projects and on an internship but want to re-familiarise myself with them to a greater extent.
- There is also some overlap with my project in the Natural Language Processing Part II unit of assessment which I am taking and I will be able to apply my knowledge to the dissertation. The scope of my dissertation, however, exceeds the course in the summarisation aspects and I will need to research a lot of domain specific aspects.
- I have read a number of relevant papers that pertain to summarisation, as well as previous entrants' entries to the Spotify Podcast Challenge. No code has yet been written for the project and I am starting from scratch.

## E.3 Substance and Structure of the Project

Extractive models were the first types of models devised in order to tackle the problem of text summarisation; key sentences within the text are extracted and added to the summary. Hence, the generated summaries consist solely of sentences from the original text.

Abstractive models identify the important sections of the text, interpret and reproduce the content in a novel way. Sentences in the summary are generated rather than extracted or simply copied from the original text.

For the dissertation, I aim to train various extractive and abstractive models on the Spotify Podcast dataset and evaluate their relative performances.

1. Preprocessing the data:
    - The dataset is very large (13GB for the non-audio and 2GB for the audio components) so there will be a good deal of filtering that needs to be done. I plan on filtering out podcasts by their transcripts to create a shorter and consequently more manageable dataset for me to train my models on.
    - I will be removing any podcast transcripts that are too long, too short and contain very many adverts. In order to handle adverts I will need to train a simple classifier on an annotated set of advert-heavy transcripts. I plan on using an SVM for this that I can implement using the Python library sklearn.
2. BART model:
    - BART is a denoising autoencoder for pretraining sequence-to-sequence models. BART is trained by adding noise to text using a random noising function and learning a model to reconstruct the original text. [2]

[2]Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension by Lewis et al (2019)

- I appreciate that training a model from scratch may be very computationally expensive and seeing as I will be balancing work for my dissertation along with my units of assessment and other university-related work, I plan to finetune an existing BART model that is pre-trained on a CNN/Daily Mail news summarization dataset [3] on the podcast transcript dataset. In order to ensure that the pre-trained model works with my implementation I will try to get similiar scores as detailed in this paper by Zheng et al (2020). [4]
- I plan to train my models on the MCS in around two days; training is expensive so I will be training for only 1 or 2 epochs in the interest of time. I plan to investigate the effectiveness of using a GPU or compute cluster for speeding this up; I plan on asking to use the HPC on the department's service level 2 account which would give me access to the GPU.
- In the case that I am unsuccessful with gaining access to the HPC, I can use Colab with the use of my Cambridge GSuite account to speed up the training of my BART model.

3. Pointer Generator model:

   - These models combine ideas from extractive and abstractive summarisation and are faster to train than purely abstractive models (in my case, the BART model). They also solve the shortcomings of abstractive models, that they often reproduce factual details incorrectly and include many repetitions.

4. Evaluation of the models

   - Metrics for evaluation:
     - Whilst the original podcast descriptions are not perfect, they serve as decent "ground truth" summaries as they are generated by the creator themselves. They may contain noise, such as the contact details of the creators as well as details of sponsorships but will contain enough of the key information to use to evaluate the auto-generated summaries.
     - I plan on using qualitative and quantitative metrics for evaluation as per the paper by Razapour et al (2020) [5]; On the quantitative side, I plan on using ROUGE-L to ascertain the longest common subsequence of words and ROUGE-N to ascertain the number of overlapping N-grams among other such scoring metrics.
     - I also plan on incorporating human evaluations in order to gain an understanding of the quality of the generated summaries compared with the episode itself. I will be asking questions that explore the coherency, conciseness and accuracy of the generated summaries. In order to do this, I will use my knowledge that I gained from Interaction Design in Part 1A and Further Human Computer Interaction in Part 1B. I will need to seek ethics approval from the department committee and have set aside a few weeks for this purpose in my timetable.

[3] https://huggingface.co/facebook/bart-large-cnn
[4] Zheng, Zhang, Wang and Fan. "A Baseline Analysis for Podcast Abstractive Summarization."
[5] Rezapour, Rezvaneh, et al. "Spotify at TREC 2020: Genre-Aware Abstractive Podcast Summarization."

5. Extensions

- Using a neural network for the preprocessor:
  Thus far I have only detailed a very basic classifier, but using a neural network for classification of advert-heavy podcast episodes has the potential to be both more accurate and increase performance as this will be introducing additional learned parameters instead of using kernels.
- Multimodal analysis:
  There is more information that could be extracted from the audio components of the datasets; for example, words and phrases that have more emphasis on them may need to be weighted more heavily when it comes to summarisation and such information could only be gleaned from an audio file. This is an area for exploration. I would be following some of the work explored by Baltrusaitis et al (2018) [6]
- Focussing on category aware summarisation:
  While genres cannot be obtained from the podcast dataset itself, I could try and extract them from RSS headers and form logical groupings of podcast genres and append them to the transcript in some way whilst fine-tuning/ training models. I may need to use a clustering algorithm, such as k-means to obtain a suitable set of genres.
- Reinforcement Learning model:
  Following the results of this paper by Paulus et al [7], I plan on training a neural network model that is trained in such a way that it combines standard supervised word prediction and reinforcement learning.
  This approach is another way to combat the shortcomings of purely abstractive models and will provide for interesting comparisons.

[6] Baltrušaitis, Tadas, Chaitanya Ahuja, and Louis-Philippe Morency. "Multimodal machine learning: A survey and taxonomy."

[7] Paulus, Xiong and Socher (2017): "A Deep Reinforced Model for Abstractive Summarization"

## E.4 Success Criteria

The project will be considered a success if, given a transcript, my summarisation system produces a summary that is shorter than the original transcript. I should have collected qualitative and quantitative evaluations that are indicative of a functional summarisation system and compared their performance on the datasets appropriately.
The success of my project is hence dependent on a few key components:

- Preprocessor:
  The preprocessor should be able to produce podcast transcripts that are devoid or light on advertisements. This can be evaluated by using standard metrics like precision and recall.
- BART model:
  The BART model should be implemented and should produce some sort of summary of a podcast given a transcript.
- Pointer Generator model:
  The Pointer Generator model should be implemented and should produce some sort of summary of a podcast given a transcript.
- Evaluating the models:
  The models should be compared against each other on certain metrics that I have specified earlier in the proposal, such as the ROUGE-n metrics.

## E.5 Timetable and Milestones

I plan to divide my time into 2 week slots, in order to allow for sufficient granularity while not committing myself to a very tight schedule.

- Slot 0- 1st October - 18th October:
  - Read papers
  - Complete project proposal draft
  - Write proposal given feedback from the proposal
  - Set up GitHub repository, install libraries
  - Deadlines: Phase 1 proposal - 5th October, 12 noon
    Draft proposal - 8th October, 12 noon
    Proposal deadline - 18th October, 5pm
  - Milestone: Proposal submission
- Slot 1- 19th October - 2nd November:
  - Familiarize myself with PyTorch, TensorFlow and any other libraries that I plan on using; I plan on working through some online tutorials
  - Implement the preprocessor; train a basic classifier and partition the dataset into training and test sets
  - Test the resulting dataset; write tests for specific cases

  - Make inquiries about the HPC and ask for permission to use it for the project
  - Milestone: Preprocessor written

- Slot 2- 3rd November- 17th November:
  - Research BART models and how they are implemented
  - Implement a BART baseline
  - Set up the project on MCS machine
  - Train model on MCS
  - If not completed, make inquiries about using the HPC.
  - Milestone: Baseline BART model written, confirm whether access to the HPC is possible or not.

- Slot 3- 18th November - 2nd December:
  - Finetune BART model; experiment with various filtering methods- random sentence filtering, truncation, TextRank, among others
  - Write scripts to evaluate the BART model
  - Milestone: BART evaluation scripts and experimentation done

- Slot 4- 3rd December- 17th December:
  - Implement Pointer Generator model
  - Train model on MCS
  - Write scripts to evaluate the model
  - Milestone: Pointer Generator evaluation scripts and experimentation done

- Slot 5- 18th December - 1st January:
  - Slack period; if not necessary, I will bring forward the evaluation of the models and look into the extension tasks ahead of time
  - Seek ethics approval from department committee
  - Milestone: All core methods implemented

- Slot 6- 2nd January - 16th January:
  - Evaluate the models based on the scripts written previously
  - If not done already, obtain the necessary approval from the department committee.
  - Milestone: Quantitative evaluation complete, approval from department committee gained.

- Slot 7- 17th January - 31th January:
  - Prepare progress report and presentation
  - Prepare for implementation of one of the extension tasks

  - Prepare questions for qualitative analysis and send to human evaluators.

- Slot 8- 1st February - 15th February:
  - Rehearse for progress report presentation
  - Receive responses from the human evaluators, aggregate and document them.
  - Implement and evaluate the first extension
  - Milestone: Complete presentation and progress report
  - Deadlines: Progress Report- 4th February, 5pm
    : Progress Report Presentation- 10th/11th February, 2pm

- Slot 9- 16th February - 2nd March:
  - Slack period; ensure that first extension is complete and outline plan for write-up before end of term
  - If I am ahead of time, I could consider implementing a second extension- potentially creating a UI could be finished quickly
  - Milestone: Complete first extension.

- Slot 10- 3rd March - 17th March:
  - Start writing up the dissertation; write the introduction and preparation chapters
  - Milestone: Draft for introduction and preparation chapters complete and sent to supervisors

- Slot 11- 18th March - 1st April:
  - Start writing the implementation chapter
  - Milestone: Draft for implementation chapter complete and sent to supervisors

- Slot 12- 2nd April - 16th April:
  - Write the evaluation chapter and ensure that I have acted on supervisors' feedback for any chapters written.
  - Milestone: Draft for evaluation chapter complete and sent to supervisors

- Slot 13- 17th April - 1st May:
  - Write the conclusions chapter and ensure that I have acted on supervisors' feedback for any chapters written
  - Milestone: Draft dissertation complete

- Slot 14- 2nd May - 13th May:
  - Slack period. Apply final touches to the dissertation and make sure that I have acted on all feedback that I have been given.
  - Milestone: Dissertation done!
  - Deadlines: Dissertation deadline - 13th May, 12 noon
    Source Code deadline - 13th May, 12 noon

## E.6 Resources Declaration

I plan to use my own personal laptop:

- A Lenovo Legion computer with an Intel Core i7 CPU running at 2.6GHz, an Nvidia RTX 2070 GPU with 8GB of VRAM, 16GB of system RAM and 1TB of SSD storage. The machine runs Windows 10.

I accept full responsibility for any damage that happens to this machine and I have got sufficient money set aside to handle any machine repairs or replacements that are necessary. I can use the MCS machines in the meantime in the event of machine failure.
In addition, I plan on asking permission to use the HPC on the department's service level 2 account; if I am successful in gaining access to the HPC then this is also a resource I will be using.
I have taken a 3-pronged approach to protect myself against errors and data loss:

- I will be pushing changes made to my code and dissertation to a GitHub repository at the end of every day that changes are made. I will also be able to use this as a version control system.
- I will ensure that the contents of my git repository and dissertation are routinely backed up onto my University OneDrive; I will set up a scheduled task on my laptop.
- I will copy the contents of my git repository and dissertation onto a USB memory stick every two weeks, at the end of every time chunk scheduled in my timetable.

In order to carry out my dissertation I need access to the Spotify Podcast dataset [8] and this has been obtained already. This was done by filling out a form on the official Challenge website [9]. These files are shared via the cloud and hence I will never lose access to them.

[8] "100,000 Podcasts: A Spoken English Document Corpus" by Ann Clifton, Sravana Reddy, Yongze Yu, Aasish Pappu, Rezvaneh Rezapour, Hamed Bonab, Maria Eskevich, Gareth Jones, Jussi Karlgren, Ben Carterette, and Rosie Jones, COLING 2020
https://www.aclweb.org/anthology/2020.coling-main.519/

[9] https://podcastsdataset.byspotify.com/