Data Science Practical

Data quality metrics for text data

Elite Master Program Data Science
Ludwig-Maximilians-Universität München
in collaboration with
BMW Group

Vladana Djakovic
Valari Pai
Ekaterina Shmaneva

Munich, May 30th, 2022

Supervised by Dr Maka Karalashvili and Dr Matthias Schubert

Abstract

While designing and producing a vehicle, different issues and defects arise both in the prototyping and production phases. All these human written defects are documented and stored in a database called Knowledge Base. Due to the complicated and time-consuming analysis of such data, the model was proposed that preprocesses the data, summarizes it and classifies it according to the given labels.

Current protocol shows how the application of the Google T5 model for summarization and DistilBERT for classification can solve the task and present the results of the implemented models.

Contents

1 Motivation

Before starting a vehicle model series production, the vehicle concept and prototype engineering are tested. Above all, the company collects test result data to track quality defects that require some rework. This data is referred to as the Prototype. Afterwards, a vehicle model goes into a series production in a plant, and, during the production phase, similar data is collected similarly for a similar purpose. The latter defects get reported as a ticket that needs to be resolved by the end of vehicle production. This data is referred to as the Production.

All quality defect recordings that were created during either of the mentioned phases contain a human, free text description. For better maintenance of these defects, a more manageable, superordinate data source is built to summarize similar quality defects in Production and Prototype. Specifically, each recording in these data should describe a prebuilt, known defect cluster. Besides similar defects, this data source should summarize similar steps conducted to fix those defects.

However, the analysis of such an amount of the human-written text data requires a high amount of humane workers and their working hours. Thus, there arose a necessity to create a model which is able to not only preprocess such data but also analyse the text of the mentioned defects, summarize them for better and easier readability and classify it.

2 Related work

Natural Language Processing (NLP) is a machine learning field that allows computers to analyze, control, and possibly create human dialect. Over the last years, it is getting more automated, allowing people to perform all kinds of tasks: from text recognition to text generation. Many new libraries and methods were created to help data scientists to proceed in this area.

Thus, Devlin et al. [1] introduced the BERT - Bidirectional Encoder Representations from Transformers - a language representation model that can perform eleven different NLP tasks (e.g. next sentence prediction, question answering or generating the summary of the written text). Different BERT models were trained with various approaches, such as diverting the number of layers, hidden units, and attention heads using the same hyper-parameters and training procedure. These tests have shown that BERT is a competitive model that is effective for both fine-tuning and feature-based approaches.

However, BERT is a technology helping to generate "contextualized" word embeddings/vectors which makes it very compute-intensive at inference time. As an answer to such limitations, many new models for optimizing the BERT were generated, for example, RoBERTa (a robustly optimized BERT approach) ([2], [3]). The main difference between the two models is that the latter was not only trained on the bigger amount of data but was also trained on the longer sequences. Moreover, tokenization is RoBERTa is performed with a byte-level Byte-Pair Encoding (BPE) encoding scheme, and its library contains 50K subword units, whereas BERT's character-level BPE only has a 30K vocabulary [4]. Being pretrained on the raw texts only, without human-created labels, RoBERTa is perfectly suited for such NLP tasks as summarisation. Singh [5] suggests using it to create an abstractive summary of the reviews that Amazon users wrote for the products they purchased.

Another example of the models capable of doing a summary are the OpenAI's GPT engines ([6], [7], [8]). Generative Pre-trained Transformer can perform different NLP tasks as well: question answering, textual entailment, text summarisation etc. without any supervised training. Also, they require exceptionally few to no examples to get the tasks and perform equivalent or even better than the state-of-the-art supervised trained models [9]. Despite GPT-2 showing performance similar to or lesser than classic models that were trained for summarisation [9], GPT-3's summarisation functionality is a great way to create summaries for books, papers, and articles [8].

Lately, transfer learning has proven to be a powerful technique in NLP. The idea behind it is to pre-train the model on a data-rich task first before fine-tuning it on a downstream task. Aiming to set a new state of the art in the field, the Google research team offered an approach to transfer learning in NLP, that was called Text-to-Text Transfer Transformer (T5) ([10], [11]). T5 model has 11 billion parameters and showed a great performance on 17 NLP tasks, for example, text classification [12], data to text generation [13] and summarisation [14]. As a matter of fact, the text-to-text architecture of the T5 made it easy to feed structured data into the model. However, in the case of bigger data, it tends to ignore some of the information in the data input. As for text classification tasks, T5 performs even better, reaching over 90% accuracy threshold, despite considering only 512 tokens of the input.

Nevertheless, the main goal of the current project is solving the task of the text summarisation, which can be performed by all of the above-mentioned models and following the classification of the generated summaries. The next sections of this report elaborate in detail on the theoretical aspects of both tasks and state the application of the Google T5 model to it.

3 Theoretical aspects

3.1 The task of summarization

According to [15], a **summary** can be defined as a text that is produced from one or more texts that contain a significant portion of the information in the original text(s), and that is no longer than half of the original text(s). Hence, training a computer to produce such a summary is called the task of **automatic summarization**.

Summarization aims to condense some text data into a shorter version while preserving most of its meaning. This ultimately saves storage and time resources that processing the long text requires. Summarization also helps to discard irrelevant information and focus on the central ideas of the text.

Generally, machine (automatic) summarization is split into two types [16]:

1. **Extractive**: Here, important text or sentences are extracted as they appear in the original document and grouped to form a concise summary. Most extractive summarization techniques focus on finding and extracting Keywords from the parent text. It can be compared to highlighting the most crucial parts of the text with a marker.

2. **Abstractive**: This approach focuses on generating summaries using the important ideas or facts that the document contains without repeating them verbatim. It is similar to the summary that a human would write after reading the text.

3.2 The task of classification

The task of **classification** can be defined as categorizing open-ended text into two or more predefined classes based on some rules or similarities between these texts. It provides valuable insights about unstructured text data as it divides them into classes.

There are three main approaches to machine-based classification tasks: [17]:

1. **Rule-based systems**: In this approach, the text is classified by using a set of linguistic rules that can be defined by the user. Usually, the rule is based on some keywords that indicate the text belonging to a particular group.

2. **Machine learning-based systems**: A machine learning algorithm learns to make classifications based on past observations. Training data with labelled examples is vital for this approach.

3. **Hybrid systems**: These are a combination of both of the above-mentioned approaches. They are useful to build classifiers for a unique task for greater precision.

3.3 Existing methods and approaches

The most common approaches are reviewed in terms of their usability for classification and summarisation in this section. All models are separated into three groups, depending on the tasks they can be performed on.

1. **Models, only used for summarization tasks**

   - *Sumy* is a library that provides a variety of algorithms for text summarization. Some of these algorithms are LexRank, Luhn, Latent Semantic Analysis (LSA), and KL-Sum. All of them are based on different concepts, which are suitable for different tasks. Sumy is also easy to use, as the algorithm can be imported without much coding or fine-tuning. However, most of the algorithms in Sumy are supposed to be used for extractive summarization [18].

2. **Models, only used for classification tasks**

   - *Naive Bayes* algorithm provides a probabilistic classifier that is based on the Bayes' Theorem. The classification is implemented by calculating the probability of each 'tag' or 'class' for the given text and then determining the label with the highest probability.
   - *Support Vector Machines (SVM)* calculate a divisionary line between two or more classes. Such a line is known as the decision boundary and determines the best result between vectors that belong to the classes and also the ones that do not. However, the main drawback of SVMs is that they perform well only when there is a limited amount of data [17].

3. **Models, used for both summarization and classification tasks**

   - *Gensim* is a python library specifically engineered for Natural Language Processing (further: NLP) tasks.
     - **Summarization:** Gensim performs extractive text summarization using the TextRank algorithm. TextRank algorithm deems the sentences that contain words that occur most frequently as significant and assigns them a 'Rank'. The sentences with the highest rank are extracted to form a summary [18].
     - **Classification:** The Gensim library provides the Doc2vec algorithm that is strong enough to perform Multi-class text classification. Doc2vec is similar to word2vec but uses a Distributed Bag of Words (DBOW) instead of Continuous Bag of Words (CBOW) or Skip-gram [19].
   - *Deep learning models (CNN and RNN):* Deep learning is a very important field of machine learning which represents multiple layered Neural networks that are designed to mimic the human brain [20]. For NLP, the most widely used Deep learning algorithms are Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). CNNs are traditionally used in computer vision tasks, however, recent research has shown that they are as well very effective on NLP tasks. RNNs are specifically designed to process sequential information.
     - **Summarization:** CNNs have mostly been implemented to only perform extractive text summarization. However, these models have a complex architecture, are highly computationally expensive, and are hard to interpret. It is also hard to implement deep bidirectionality using CNNs. https://leolaugier.github.io/doc/summarization.pdf

Recurrent Neural Networks (RNN) have also been used to perform extractive text summarization with state-of-the-art performance. GRU and LSTM models have an easy-to-interpret approach but can only deliver high performance in specific cases. The main drawback of RNN models is that they cannot handle long-term dependencies. Cited from NLP lecture

  - **Classification:** CNNs can be used for classification by utilizing a feature that is applied to words or n-grams to extract high-level features [20]. RNNs are also effective in performing classification as they have the ability to memorize the previous output and use that information to base the next one [17].

- *BERT and BERT-based models:* BERT is a bidirectional transformer-based model which was implemented to overcome the drawbacks of the RNN models. BERT (Bidirectional Encoder Representations from Transformers) [1] is a pre-trained model which can be easily fine-tuned to perform multiple tasks. BERT was a revolutionary model which provided a strong architectural base for many other models. These models mostly focus on improving BERT's performance or making it more efficient. Some examples of such models are ALBERT (A smaller model with stronger performance) [21], RoBERTa (A larger model with more parameters aimed at making BERT more robust) [3], and DistilBERT (A distilled version of BERT that is faster, smaller and lighter) [22], etc.
  Since the fine-tuning of BERT and Co to perform any NLP tasks is unchallenging, they can perform both classification and summarisation. BERT is also very effective in performing abstractive summarization.

- *T5:* Google's text-to-text transfer transformer model is trained end-to-end with a text string as the input and a modified text string as the output. This gives the T5 model an advantage over BERT-based models as the latter only returns a class label.
  The T5 model is used to perform multiple NLP tasks with state-of-the-art performance including abstractive summarization. This is a pre-trained model which is trained on the unlabelled large text corpus called C4 (Colossal Clean Crawled Corpus) using deep learning [14].
  There are five different versions of the pre-trained T5 model available on HuggingFace: T5 depending on the size of the model. The smallest is the "T5-small" with 60 million parameters, whereas the largest, "T5-11B", has 11 billion parameters.
  T5 is implemented using HuggingFace transformers and can be fine-tuned to the required NLP task. So, it can perform both Classification and Summarization tasks.

- *GPT models:* OpenAI's GPT (Generative Pre-trained Transformer) is one of the most well-known NLP models out there. The latest version, GPT-3, has 175 billion parameters that give the model a tremendous amount of power. GPT-3 can be used for all sorts of NLP tasks and outperforms many state-of-the-art models [8]. However, GPT-3 is not open-sourced and, hence, can only be used via an API after registration.

- *XLNet:* The XLNet model can be interpreted as a modification of the BERT model. It is a bidirectional transformer-based model which is pre-trained in a regressive manner, similar to the GPT family of models. It comes in two versions, which differentiate in size: XLNet-base-cased and XLNet-large-cased. Because of its size, XLNet is very expensive to evaluate the SotA (State of the Art) results of the XLNet-large model. However, it generally gives very good results on downstream language tasks like question answering, sentiment analysis, etc [23]. Though, when it comes to summarization, it is outperformed by T5 [6].

3.4 Model choice

Over the past few years, transfer learning has led to a new wave of state-of-the-art results in natural language processing. Transfer learning's effectiveness comes from pre-training a model on abundantly available unlabeled text data with a self-supervised task, such as language modelling or filling in the missing words. After that, the model can be fine-tuned on smaller labelled datasets, often resulting in a better performance than training on the labelled data alone. The recent success of transfer learning was ignited in 2018 by GPT, ULMFiT, ELMo, and BERT, and 2019 saw the development of a huge diversity of new methods like XLNet, RoBERTa, ALBERT, Reformer, and MT-DNN. The rate of progress in the field has made it complicated to evaluate which improvements are most meaningful and how effective they are when combined.

3.4.1 Summarization

First research showed that the best model (from BERT-styled models) for summarization is RoBERTa. RoBERTa is an encoder model similar to BERT, but it uses dynamic MASKing. So, RoBERTa sees the same sequence masked differently, unlike BERT who sees the MASKed sequence only once. It also completely discards the NSP objective and uses a much larger corpus (160GB) during pre-training instead. This provides RoBERTa with much better results than BERT and XLNet model [3].

After implementation of RoBERTa authors discovered that this model was not the best suitable for the task. RoBERTa is just an encoder-based model and, thus, does not perform well on summarization tasks. Research showed that picking either an encoder-decoder based model or only a decoder based model will provide better results for summarization.

Wanting to explore the limits of Transfer Learning, researchers at Google wanted to create a unique model which could be applied to many NLP tasks such as summarization, translation, questions, and answers. The model was named Text-To-Text Transfer Transformer (T5). Unlike BERT, which had only encoder blocks, T5 uses both encoder and decoder blocks. Moreover, T5 does not output a label or a span of the input to the input sentence, and the output is a text string as well. This reason makes the T5 model more suitable for summarization tasks than any BERT-styled model. Due to the lack of computational resources, the authors decided to confine to the "T5-small" version, which was as well pre-trained on a multi-task mixture of unsupervised and supervised tasks, and performs not worse, than its extended variations [24].

3.4.2 Classification

Due to the benefits of transfer learning, the authors decided to implement and fine-tune a pre-trained model. Since classification and sentiment analysis is a task much simpler than summarization, BERT-style models are still a good choice. The developers of this project were limited in computational resources, so it was decided to implement the DistilBERT model. DistilBERT is a much smaller, faster and cheaper model compared to BERT and has provided SOTA results for classification tasks. DistilBERT is created by distilling the BERT base model. As a result, it has about 40% fewer parameters than BERT which gives it the ability to run 60% faster. Despite this, DistilBERT is capable of retaining around 95% of BERT's performance. The authors of DistilBERT have tried to minimize inductive biases which large models usually learn to manage during pretraining by using a triple loss approach which focuses on language modelling, distillation and cosine-distance losses. Further, this model can also be found on HuggingFace just like BERT giving it the same level of flexibility. Because of the above-stated reasons, DistilBERT is an obvious choice for the task of classification under this particular scenario. [22].

Implementation aspects and computational results are as well provided in further chapters of this work (c.f. Sec. 5.2 for the details on implementation and Sec. 5.3 for the performance analysis).

4 Background & prerequisites

4.1 Project goals

The main goal of the project is to derive reasonable quality metrics for text data in the data sources, analyze the free text description data, and generate and analyze a summary of it.

Normally, projects of such scope can have two target audiences: the first one is a *Data Steward*, who takes ownership of the data, works with the business to define the programme's objectives [25], and, thus, not necessarily equipped with a Data Science background. The second one is a *Data Scientist* - analytical data expert, who has technical skills to solve complex problems [26]. This implies that every task within the project should be easily understood by persons without a Data Science background as well.

One of the project tasks is the derivation of the suitable metrics from the data, the choice of which strongly depends on the available data. Hence, metric derivation and output of the expected results should be included in the list of goals as well.

Due to several data security issues, analyzing and handling the initial data was impossible. Consequently, the research for the open-source data, which has the most similar structure, was conducted. This resulted in the use of the Amazon Product Reviews dataset, which has different categories and a great number of reviews that could not be processed with existing resources. Thereby, the authors narrowed the data to utilizing the Quality Food reviews. This data set was suitable to obtain a good performance of the model and had a diversity, that was most similar to the original data, for which this project was designed to handle. Based on the new data, a set of new goals was defined, including data preprocessing, summarization, and analyzing the goodness of the summary.

4.2 Data overview

Amazon Product Review dataset is a publicly available dataset [27], which contains 568.454 reviews. The data is stored within 10 columns: *Id, ProductId, UserId, ProfileName, HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary, and Text* (Fig. 1).

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

Figure 1: A preview of data

To proceed with the further analysis, first of all, all reviews, that do not have a value, are dropped. This cleans the dataset up to 568.427 reviews.

Moreover, not all columns are needed for summarization tasks: only columns *Summary* and *Text* are kept, whereas other columns are not important and are discarded. For more precise summaries, stop words were removed using the additional filter on the data. Another filter helped to exclude all reviews that were too long (reviews longer than 512 tokens).

The distribution analysis of tokens in columns *Text* and *Summary* within the remaining data can help to get a better understanding of the data (Fig. 2).

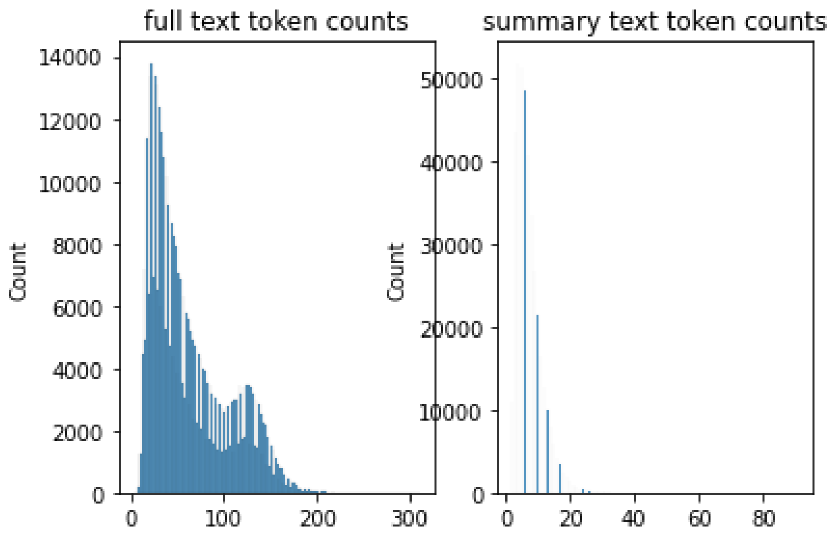

Figure 2: Distribution of the number of tokens in *Text* and *Summaries*

As a baseline model, a function, the Git repository for which was linked in the code file, was used to untokenize the reviews back as the text. Additionally, due to the lack of computational resources which were at their disposal, the authors could not perform the training task on the whole dataset. Hence, a larger subsamples of the reviews of different size were was used.

Similarly, for classification task, authors only kept columns *Score, Summary*, and *Text*. Column *Score* refers to the rating (on a scale of 1 to 5) provided by the reviewer for the variety of food products, that amazon offers. This *Score* was then used to calculate the *Sentiment*: a Boolean value (positive or negative) to indicate the sentiment of the review based on the rule: if the score is greater than or equal to three, positive, otherwise negative.

4.3 Data processing and baseline models

Summarization and classification tasks can be performed with various models, most of which are available in the Huggingface library in Python. Huggingface library is specially designed for NLP Transformers implementation, and it supports other widely used Python libraries. As a summarization model, the authors used the Google T5 model, whereas for the classification task, the DistilBERT model was chosen. Both models can be imported from the Transformers package. Additionally, to enable the training of the model, data needs to be encoded in an appropriate way using a predefined Tokenizer.

In Python, the T5 model is implemented in several sizes: *t5-small, t5-base, t5-large, t5-3b and t5-11b*. The difference between models is illustrated in Fig. 3. As a consequence of available computation power, we have implemented a t5-small model for generating summaries of Reviews.

Model size variants

Model	Parameters	# layers	d_{model}	d_{ff}	d_{kv}	# heads
Small	60M	6	512	2048	64	8
Base	220M	12	768	3072	64	12
Large	770M	24	1024	4096	64	16
3B	3B	24	1024	16384	128	32
11B	11B	24	1024	65536	128	128

Figure 3: T5 model size variants [11]

For the T5-small model, the authors have split the dataset into train, validation, and test datasets in the ratio of 80-10-10, which is one of the most used techniques [28]. Afterwards, each dataset was encoded using **T5TokenizerFast** from the Transformers package to suit the desired model input. Moreover, the maximum length of tokens was set to 512 tokens for *Text* and 128 for *Summaries*. All larger *Text* and *Summary* were cut after respecting the maximum token length. More information about this is provided in Sec. A.2.

Similar processing was utilized while implementing the Classification model. Here, instead of the T5, the DistilBERT model was used, which is as well importable via the Transformers package. The *Text* field is used as input with *Sentiment* as a label to train the model. Different from the split of the dataset for the T5 summarization model, here authors used a stratified split using the *train_test_split* function from sklearn library. To train and fit the model, the text input was first encoded using the pre-trained DistilBert Tokenizer. This is done to remove punctuation splitting and word pieces. The authors then proceed to fit a pre-trained DistilBert Classification model on the prepared dataset.

5 Implementation storyline

5.1 Project storyline

The project started on the the 18th of October 2021 with the Kick-off from the BMW side. The developers were made familiar with the task and dataset to use and suggested starting the research on the possible methods. However, as stated in the 4.1, due to various issues with the access to the data and data security, there arose the necessity to conduct additional research on the plausible for the task of open-source data. That affected the timeline and the scope of the project (Fig. 4).

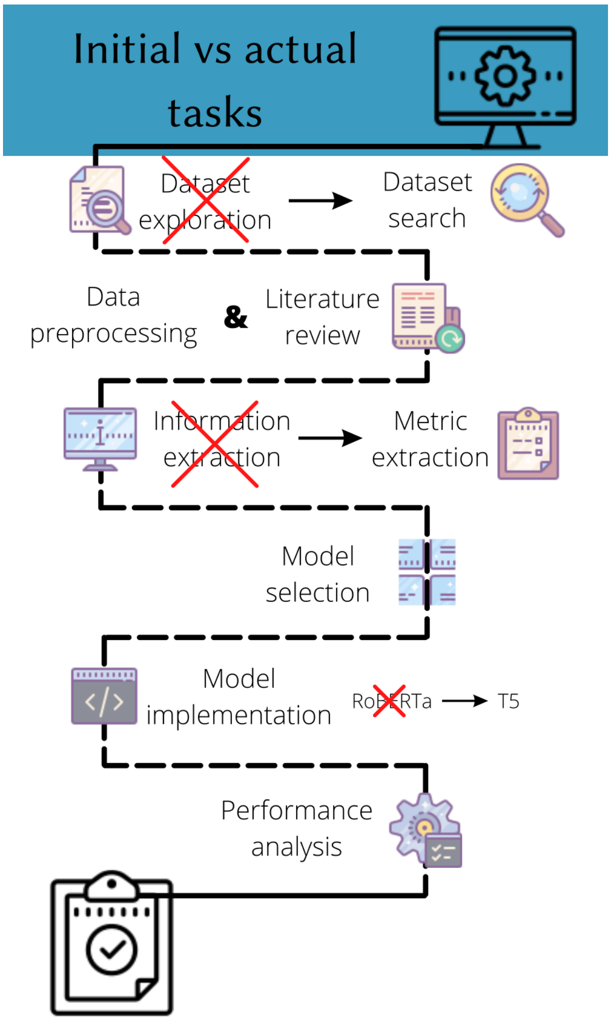

Figure 4: The scope of the project, including tasks, changed due to unexpected issues

All further tasks, displayed within the scope are described in more detail consecutively in this section.

Metric selection and extraction As a result of the research, the Amazon Product Reviews datasets [27] were found and examined. The first focus fell on the Musical

Instruments reviews and product metadata. Being compressed in the JSON format, the data contained the following dimensions: *reviewerID* (ID of the reviewer), *asin* (ID of the product), *reviewerName* (name of the reviewer), *helpful* (helpfulness rating of the review), *reviewText* (text of the review), *overall* (rating of the product), *summary* (summary of the review, written by user), *unixReviewTime* (time of the review (unix time)), *reviewTime* (time of the review (raw)).

Developers proposed using three metrics, provided in the dataset:

1. **helpfulness** analysis as a first layer rule-based evaluation. If the helpfulness exceeded some preset threshold, **reviewText** analysis is performed.

2. **reviewText** analysis should be performed according to some characteristics of the text, such as length, most occurring words, repetitions, the mood of the review, and product names within it.

3. **overall** score, given by the user, should be consistent with the mood of the review itself.

However, during the approach discussion, it was decided to reduce the analysis to the **reviewText** dimension, as it is the most important metric for the initial task. In parallel, keyword search and summarization techniques were researched and evaluated.

Model selection The model choice is described more precisely in the Sec. 3.4. Moreover, together with the model selection, a bigger and more applicable dataset was found for a better analysis. The new dataset was as well containing the Amazon Product reviews, yet still, a different preprocessing was needed (c.f. Sec. **??**).

At the same time, literature research and review were conducted (c.f. Sec. 2). After the above-mentioned steps project entered its practical phase, including the tasks of metrics analysis for the new dataset, evaluating existing summarization and classification techniques and corresponding Python packages.

Model implementation Implementation aspects are discussed in Sec. 5.2 of the current report. The first model that was selected for implementation was RoBERTa [4]. As mentioned in Sec. 3.4.1, it is an encoder model that suffers from some issues with weight assignment to the decoder. Thus, although the model was correctly implemented and adapted to chosen data, its training was performed incorrectly. Hence, the T5 model was chosen as it is more robust and easier in issue handling.

Performance analysis All outputs and their interpretation are thoroughly discussed in the Sec. 5.3.

Furthermore, the project is tied to time, which made time planning crucial. Though, as discussed in the paragraphs above and the Sec. 3.4, due to unexpected issues with the RoBERTa model, the authors were forced to change the model used for summarisation. Consequently, the deadlines of the project had to be moved to a later period, resulting in the new timeline (Fig. 5).

Needless to say, every complicated model requires a lot of work and bug fixing. Hence, the authors of this report came across several issues during the project that as well

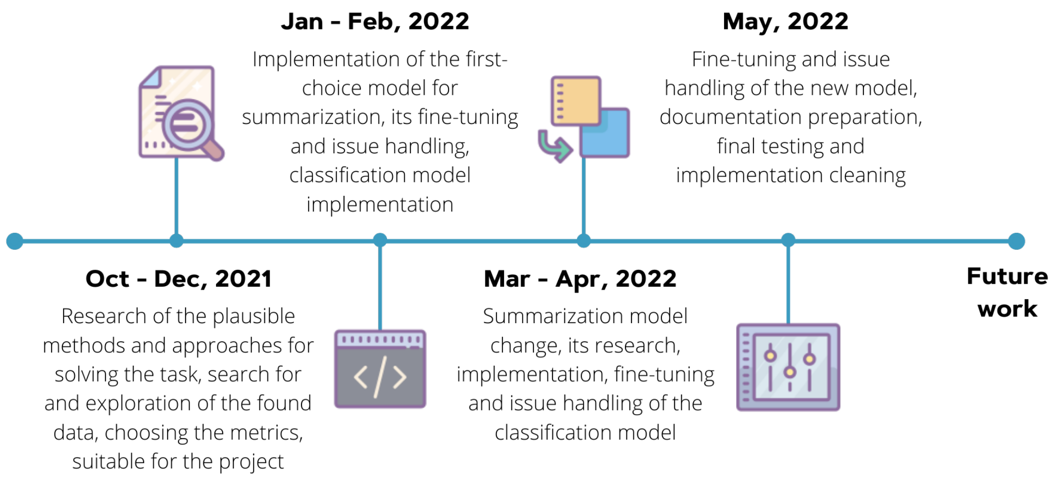

Figure 5: The updated timeline of the finalized project

affected the deadlines and resulted in the prolongation of the project. The most crucial and relevant of them are shown in Fig. 6.

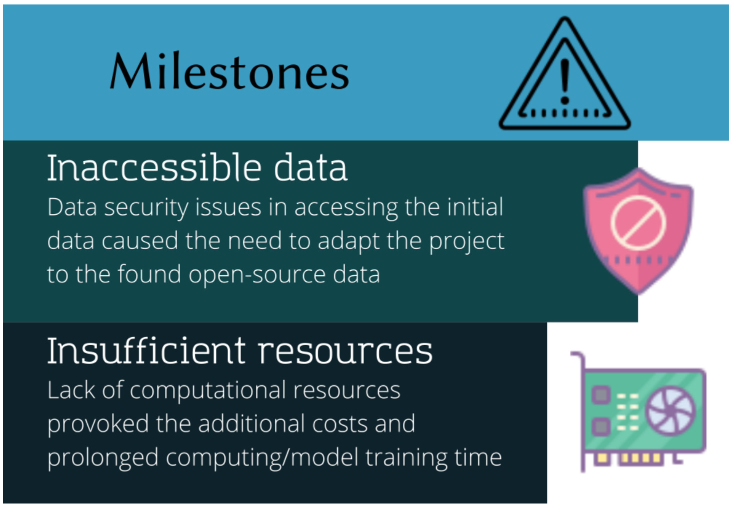

Figure 6: The most troublesome aspects of the project, that resulted in the necessity to expand the deadlines

5.2 Implementation aspects and set up

Project set up The following set up was used to proceed with the described models:

1. **Environment:** Python ver. 3.8.5 and above
2. **Computational system:** Google Colab (Pro)

3. **Processing machine:** Google GPU Accelerator
4. **:**

Summarization implementation Section 3.1 of the current document refers to the different types of summarizations. Initially, the standard BERT Library [1] was chosen. Although, for easier and faster computation, its distilled modification (DistilBERT [22]) was used. Additionally, the robustly optimized BERT approach, RoBERTa [3], was applied for creating summarisations.

The vast majority of the NLP models process the data on the token level. Thus, tokenization is a crucial step while conducting such tasks as summarisation. However, during the training, RoBERTa showed some shortcomings, namely, the weakness of its tokenizer: some weights of the model checkpoint were not used when initializing the model. Hence, it could not learn anything during training. The authors performed further research and after the trial-test phase, shifted their attention to the application of the Text-to-Text Transfer Transformer (T5). Despite being bulkier than its predecessor, the T5 model has proven to be more accurate in performing summarization tasks. Additionally, its tokenizer is more robust. Thus, the in-built *T5TokenizerFast* module was used in the model:

```
# Initialising tokenizer
modelName="t5-small"
tokenizer = T5TokenizerFast.from_pretrained(modelName)

text_token_counts =[]
summary_token_counts = []
# Checking distribution of tokens in columns
# Text and Summary to get feeling about data distribution
for _,row in train_data.iterrows():
  #keeping first 512 tokenized values
  text_token_count = len(tokenizer.encode(row["Text"][:512]))
  #NOTE this lenght is not the same as lenght of Text of review
  text_token_counts.append(text_token_count)

  summary_token_count = len(tokenizer.encode(row["Summary"]))
  summary_token_counts.append(summary_token_count)
```

Classification implementation Another goal of the project was to classify generated summaries after they were created. Another model was created for that part, however, in this case, remaining with the use of BERT (precisely, DistilBERT) and applying NLTK library additionally proved itself suitable.

Two classes were defined: *Positive* and *Negative*, meaning the level of customers' satisfaction with the reviewed product. The analysis was done for both:

1. the initial text of the review

```python
#Classifying the text into sentiment classes
for i in range(0,len(df)):
    predict_input_text = tokenizer.encode(df['Text'][i],
                                          truncation=True,
                                          padding=True,
                                          return_tensors="tf")
    tf_output_text = loaded_model.predict(predict_input_text)[0]
    tf_prediction_text = tf.nn.softmax(tf_output_text, axis=1)
    labels = ['Negative','Positive']
    label_text = tf.argmax(tf_prediction_text, axis=1)
    label_text = label_text.numpy()
    df["Sentiment_text"][i] = (labels[label_text[0]])
```

2. and a newly generated summary.

```python
#Classifying the generated summaries into sentiment classes
for i in range(0, len(df)):
    predict_input_sum = tokenizer.encode(df['Generated_summary'][i],
                                         truncation=True,
                                         padding=True,
                                         return_tensors="tf")
    tf_output_sum = loaded_model.predict(predict_input_sum)[0]
    tf_prediction_sum = tf.nn.softmax(tf_output_sum, axis=1)
    labels = ['Negative','Positive']
    label_sum = tf.argmax(tf_prediction_sum, axis=1)
    label_sum = label_sum.numpy()
    df["Sentiment_summary"][i] = (labels[label_sum[0]])
```

During the training, the performance of the model was analysed and resulted in over 90% accuracy. Moreover, the results between classified texts and summaries were compared and the error in classes for generated summary and text input did not exceed 10%.

5.3 Performance analysis

To analyze the overall performance of the models and to understand how far the authors managed to achieve the goal of the project, several metrics were chosen and calculated during the project for each task:

1. **Running walltime ???**

2. **Used memory ???**

3. **Accuracy** is the number of correctly predicted data points out of all the data points.

4. **Validation** and **Training loss** that describe the performance of the model, indicating how well it is fitting the training and the new data correspondingly.

5. **Rouge score** stands for Recall-Oriented Understudy for Gisting Evaluation [29]. It works by comparing an automatically produced summary or translation against a set of reference summaries (typically human-produced). There are several calculation techniques, but authors use the *ROUGE-1* (overlap of unigrams between the system summary and reference summary), *ROUGE-2* (overlap of bigrams between the system and reference summaries) and *ROUGE-L* (longest matching sequence of words using LCS).

6. **Classification error** is calculated as a percentage of the generated summaries, that were classified differently, than corresponding initial review text (i.e. if the summary is classified as negative whereas the review text was marked as positive, the counter increases).

Performance metrics describing the behaviour of the model in terms of the use of computational resources * Summarization - Running walltime - Used memory
* Classification - Running walltime - Used memory

Performance metrics describing the behaviour of the model in terms of the achieved results * Summarization - Accuracy - Train loss - Validation loss - Rouge score
* Classification - Accuracy - Train loss - Validation loss - Classification error

6 Future work

1 Further classification and/or clusterization of the data (based on the information, that summary contains) 2 Score prediction (very good, very bad, neutral) 3 Further fine-tuning for better summary

7 Conclusion

A concise summary of contents and results

8 References

[1] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv:1810.04805. type: article. arXiv, May 24, 2019. DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805[cs]. URL: http://arxiv.org/abs/1810.04805 (visited on 05/22/2022).

[2] Aastha Singh. *Evolving with BERT: Introduction to RoBERTa*. Analytics Vidhya. July 9, 2021. URL: https://medium.com/analytics-vidhya/evolving-with-bert-introduction-to-roberta-5174ec0e7c82 (visited on 05/22/2022).

[3] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv:1907.11692. type: article. arXiv, July 26, 2019. arXiv: 1907.11692[cs]. URL: http://arxiv.org/abs/1907.11692 (visited on 05/22/2022).

[4] Rohan Jagtap. *RoBERTa: Robustly Optimized BERT-Pretraining Approach*. DataSeries. Aug. 19, 2020. URL: https://medium.com/dataseries/roberta-robustly-optimized-bert-pretraining-approach-d033464bd946 (visited on 05/22/2022).

[5] Anubhav. *Step by Step Guide: Abstractive Text Summarization Using RoBERTa*. Medium. Dec. 20, 2020. URL: https://anubhav20057.medium.com/step-by-step-guide-abstractive-text-summarization-using-roberta-e93978234a90 (visited on 05/22/2022).

[6] Manmohan Singh. *Summarize Reddit Comments using T5, BART, GPT-2, XLNet Models*. Medium. Jan. 4, 2021. URL: https://towardsdatascience.com/summarize-reddit-comments-using-t5-bart-gpt-2-xlnet-models-a3e78a5ab944 (visited on 05/22/2022).

[7] Sukanya Bag. *Text Summarization using BERT, GPT2, XLNet*. Analytics Vidhya. Apr. 26, 2021. URL: https://medium.com/analytics-vidhya/text-summarization-using-bert-gpt2-xlnet-5ee80608e961 (visited on 05/22/2022).

[8] kdnuggets. *Summarization with GPT-3*. KDnuggets. Section: Products and Services. URL: https://www.kdnuggets.com/summarization-with-gpt-3.html/ (visited on 05/22/2022).

[9] Priya Shree. *The Journey of Open AI GPT models*. Walmart Global Tech Blog. Nov. 10, 2020. URL: https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2 (visited on 05/25/2022).

[10] Maria Yao. *10 Leading Language Models For NLP In 2021*. TOPBOTS. May 11, 2021. URL: https://www.topbots.com/leading-nlp-language-models-2020/ (visited on 05/22/2022).

[11] Qiurui Chen. *T5: a detailed explanation*. Analytics Vidhya. June 11, 2020. URL: https://medium.com/analytics-vidhya/t5-a-detailed-explanation-a0ac9bc53e51 (visited on 05/22/2022).

[12] pedrormarques. *Fine tuning a T5 text-classification model on colab*. pedrormarques. Oct. 21, 2021. URL: https://pedrormarques.wordpress.com/2021/10/21/fine-tuning-a-t5-text-classification-model-on-colab/ (visited on 05/24/2022).

[13] Mathew Alexander. *Data to Text generation with T5; Building a simple yet advanced NLG model*. Medium. Apr. 10, 2021. URL: https://towardsdatascience.com/data-to-text-generation-with-t5-building-a-simple-yet-advanced-nlg-model-b5cce5a6df45 (visited on 05/24/2022).

[14] turbolab. *Abstractive Summarization Using Google's T5*. Turbolab Technologies. Section: Technology. Oct. 4, 2021. URL: https://turbolab.in/abstractive-summarization-using-googles-t5/ (visited on 05/22/2022).

[15] Eduard Hovy. *Text Summarization*. The Oxford Handbook of Computational Linguistics. ISBN: 9780199276349. Jan. 13, 2005. DOI: 10.1093/oxfordhb/9780199276349.013.0032. URL: https://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199276349.001.0001/oxfordhb-9780199276349-e-32 (visited on 05/22/2022).

[16] Praveen Dubey. *Understand Text Summarization and create your own summarizer in python*. Medium. Dec. 23, 2018. URL: https://towardsdatascience.com/understand-text-summarization-and-create-your-own-summarizer-in-python-b26a9f09fc70 (visited on 05/29/2022).

[17] monkeylearn. *Text Classification: What it is And Why it Matters*. MonkeyLearn. URL: https://monkeylearn.com/text-classification/ (visited on 05/29/2022).

[18] Shrivar Sheni. *Text Summarization Approaches for NLP - Practical Guide with Generative Examples*. Machine Learning Plus. Oct. 24, 2020. URL: https://www.machinelearningplus.com/nlp/text-summarization-approaches-nlp-example/ (visited on 05/29/2022).

[19] Susan Li. *Multi-Class Text Classification with Doc2Vec & Logistic Regression*. Medium. Dec. 4, 2018. URL: https://towardsdatascience.com/multi-class-text-classification-with-doc2vec-logistic-regression-9da9947b43f4 (visited on 05/22/2022).

[20] Inés Roldós. *Go-to Guide for Text Classification with Machine Learning*. MonkeyLearn Blog. Section: Text Classification. Mar. 2, 2020. URL: https://monkeylearn.com/blog/text-classification-machine-learning/ (visited on 05/22/2022).

[21] Zhenzhong Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. arXiv:1909.11942. type: article. arXiv, Feb. 8, 2020. DOI: 10.48550/arXiv.1909.11942. arXiv: 1909.11942[cs]. URL: http://arxiv.org/abs/1909.11942 (visited on 05/22/2022).

[22] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. arXiv:1910.01108. type: article. arXiv, Feb. 29, 2020. DOI: 10.48550/arXiv.1910.01108. arXiv: 1910.01108[cs]. URL: http://arxiv.org/abs/1910.01108 (visited on 05/22/2022).

[23] Zihang Dai. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. original-date: 2019-06-19T08:16:46Z. May 22, 2022. URL: https://github.com/zihangdai/xlnet (visited on 05/22/2022).

[24] Zhanpeng Wang. "Smart Auto-completion in Live Chat Utilizing the Power of T5". PhD thesis.

[25] experian. *What is a Data Steward? — Experian Business.* URL: https://www.experian.co.uk/business/glossary/data-steward/ (visited on 05/29/2022).

[26] SAS-institute. *What is a data scientist?* URL: https://www.sas.com/en_us/insights/analytics/what-is-a-data-scientist.html (visited on 05/29/2022).

[27] Amazon. *Amazon review data.* URL: https://jmcauley.ucsd.edu/data/amazon/ (visited on 05/22/2022).

[28] Rachel Draelos. *Best Use of Train/Val/Test Splits, with Tips for Medical Data.* Glass Box. Sept. 15, 2019. URL: https://glassboxmedicine.com/2019/09/15/best-use-of-train-val-test-splits-with-tips-for-medical-data/ (visited on 05/29/2022).

[29] Kavita Ganesan. *An intro to ROUGE, and how to use it to evaluate summaries.* freeCodeCamp.org. Jan. 26, 2017. URL: https://www.freecodecamp.org/news/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840/ (visited on 05/22/2022).

A Appendix

A.1 The list of the Python packages, used in the project

This section contains information on all the libraries, modules and other packages, that were used for implementation of the tasks of the project. Corresponding documentation can be found in the linked references.

Name of the library	Required for the task of	Reference	Commentar (version)
AdamW	summarization	Pytorch: AdamW	
dataclass	summarization	Dataclasses: dataclass	
DataLoader	summarization	Pytorch: DataLoader	
Dataset	summarization	Pytorch: Dataset	
DistilBertTokenizer Fast	classification	HuggingFace: DistilBert	
field	summarization	Dataclasses.field	
logging	summarization	logging	
matplotlib.pyplot	summarization	Pyplot	
ModelCheckpoint	summarization	tf: ModelCheckpoint	
nltk	classification, summarization	NLTK	
numpy	classification, summarization	NumPy	1.19.5 (classification)
Optional	summarization	Typing.optional	
pandas	classification, summarization	Pandas	
pytorch_lightning	summarization	PyTorch: lightning	
re	classification, summarization	Regular Expressions	
rouge_score	summarization	Rouge score	
sacremoses	classification	PyPi: sacremoses	0.0.45
seaborn	summarization	Pydata: seaborn	
stopwords	summarization	nltk: stopwords	
StratifiedShuffleSplit	classification	sklearn: Str. Shuffle Split	
T5ForConditional Generation	summarization	HuggingFace: T5	
T5TokenizerFast	summarization	HuggingFace: T5 Tokenizer	
tensorboard	summarization	tf: TensorBoard	
TensorBoardLogger	summarization	Pytorch: tf Board logger	
tensorflow	classification, summarization	Tensorflow	2.7.0 (classification)
tensorflow_datasets	classification	tf: datasets	
TFDistilBertFor SequenceClassification	classification	HuggingFace: DistilBert	

torch	summarization	Pytorch	
train_test_split	classification	sklearn: train$_test_split$	
Trainer	summarization	Transformers: Trainer	
TrainingArguments	summarization	Transformers: TrainingArgs	
transformers	classification, summarization	HuggingFace: Transformers	4.5 (summarization), 4.7.0 (classification)
viewitems	classification	six: viewitems dictionary	
word_tokenize	summarization	nltk: word$_tokenize$	

A.2 Implementation code

Current section contains the whole implementation code of the project, separated into the tasks: **Summarization implementation** (Sec. A.2.1) and **Classification implementation** (Sec. A.2.2)

A.2.1 Summarization

```
# -*- coding: utf-8 -*-
"""T5Summarization.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1kYVQqol5iIwEye1nm5cXZ27IkIMk1K
"""

#Preinstalling necessary libraries / specificated versions are necessary tc

!pip install --quiet transformers==4.5

!pip install --quiet rouge_score

!pip install --quiet pytorch-lightning

!pip install --quiet tensorflow
!pip install --quiet tensorboard
!pip install --quiet nltk

#importing necessary libraries and packages
import json
import pandas as pd
import numpy as np
import logging
import torch
from torch.utils.data import DataLoader,Dataset
import pytorch_lightning as pl
import matplotlib.pyplot as plt
import seaborn as sns


from dataclasses import dataclass, field
from typing import Optional
#imoporting tokenizer and T5 model
from transformers import T5ForConditionalGeneration, T5TokenizerFast, AdamW


from pytorch_lightning.callbacks import ModelCheckpoint
```

```
from pytorch_lightning.loggers import TensorBoardLogger
from rouge_score import rouge_scorer

#packages and libraries for removing stopwords
import re
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

#Importing Google Drive for reading the data
from google.colab import drive
drive.mount('/content/drive')

#Reading and clearing the data
#Vladana PATH
df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Reviews.csv", engine


#Vallari PATH
#df=pd.read_csv("/content/drive/MyDrive/Reviews.csv", engine="python", erro

#Katja PATH
#df=pd.read_csv("/content/drive/MyDrive/DS Practical/Reviews.csv", engine="

df.drop(columns=['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNu
#print("Before",len(df))
df = df.dropna()
#print("Data size:",len(df))
df.head()

#Shortening the data for testing purposes (remove the whole cell for full t
df1=df.loc[1:2000]
#df.shape
#df1.shape
print("Data size:",len(df1))

tokens_wo_stopwords[:512]

#Untokenize function from
#https://github.com/commonsense/metanl/blob/master/metanl/token_utils.py
def untokenize(words):
    """
    Untokenizing a text undoes the tokenizing operation, restoring
    punctuation and spaces to the places that people expect them to be.
    Ideally, `untokenize(tokenize(text))` should be identical to `text`,
    except for line breaks.
```

```
    """
    text = ' '.join(words)
    step1 = text.replace("`` ", '"').replace(" ''", '"').replace('. . .', 
'...')
    step2 = step1.replace(" ( ", " (").replace(" ) ", ") ")
    step3 = re.sub(r' ([.,:;?!%]+)([ \'"`])', r"\1\2", step2)
    step4 = re.sub(r' ([.,:;?!%]+)$', r"\1", step3)
    step5 = step4.replace(" '", "'").replace(" n't", "n't").replace(
         "can not", "cannot")
    step6 = step5.replace(" ` ", " '")
    return step6.strip()

text = df1['Text']

#Convert text to lowercase and split to a list of words
tokens=[]
for i in range(len(text)):
  oneRow=text.iloc[i]
  tokens.append(word_tokenize(oneRow.lower()))



#Remove stop words
english_stopwords = stopwords.words('english')
tokensWoStopwords=[]

for i in range(len(tokens)):
  tokens_wo_stopwords = [t for t in tokens[i] if t not in english_stopwords
  #print(len(tokens_wo_stopwords)
  tokensWoStopwords.append(untokenize(tokens_wo_stopwords[:512]))

#print(len(tokens_wo_stopwords))

len(i.split(' '))

i

for i in tokensWoStopwords:
  if len(i.split(' '))>512:
    print(i)
    print('-------------')

#replacing Text with Text without stopwords
df1['Text']=tokensWoStopwords
df1=df1.reset_index(drop=True)
df1.head

#Shortened dataset split into train , validation and test dataset
```

```
n_train = int(np.round(df1.shape[0]*0.8))
n_val = int(np.round(df1.shape[0]*0.1))
n_test = int(np.round(df1.shape[0]*0.1))
train_data=df1.loc[:n_train]
val_data=df1.loc[n_train:n_train+n_val]
test_data=df1.loc[n_train+n_val:n_train+n_val+n_test]


#Full dataset split
#n_train = int(np.round(df.shape[0]*0.8))
#n_val = int(np.round(df.shape[0]*0.1))
#n_test = int(np.round(df.shape[0]*0.1))
#train_data=df.loc[:n_train]
#val_data=df.loc[n_train:n_train+n_val]
#test_data=df.loc[n_train+n_val:n_train+n_val+n_test]

#Checking how dataset is splitted
#train_data.shape, test_data.shape, val_data.shape

#Creating dataset shape for new T5 model for summarisation
class SummaryDataset (Dataset):
  def __init__ (
      self,
      data: pd.DataFrame,
      tokenizer: T5TokenizerFast, #initializing tokenizer
      text_max_token_len: int = 512, #setting maximum lenght of tokens for
      summary_max_token_len: int = 128 #setting maximum lenght of tokens fo
      ):
    self.tokenizer = tokenizer
    self.dataF = data
    self.text_max_token_len = text_max_token_len
    self.summary_max_token_len = summary_max_token_len

  def __len__(self):
    return len(self.dataF)

  def __getitem__(self, index: int):
    data_row = self.dataF.iloc[index]

    text = data_row["Text"]
    #encoding Text value to be suitable for pretrained T5 model
    text_encoding = tokenizer(
        text,
        max_length=self.text_max_token_len,
        padding="max_length",
        truncation=True,
        return_attention_mask=True,
        add_special_tokens=True,
```

```
        return_tensors="pt"

    )
    #encoding Summary value to be suitable for pretrained T5 model
    summary_encoding = tokenizer(
        data_row["Summary"],
        max_length=self.summary_max_token_len,
        padding="max_length",
        truncation=True,
        return_attention_mask=True,
        add_special_tokens=True,
        return_tensors="pt"

    )

    labels = summary_encoding["input_ids"]
    labels[labels==0]=-100

    return dict(
        text=text,
        summary=data_row["Summary"],
        text_input_ids=text_encoding["input_ids"].flatten(),
        text_attention_mask=text_encoding["attention_mask"].flatten(),
        labels=labels.flatten(),
        labels_attention_mask=summary_encoding["attention_mask"].flatten()
    )

# encoding train, validation and test dataset to desred input of model
class SummaryDataModule(pl.LightningDataModule):
  def __init__(
    self,
    train_df: pd.DataFrame,
    test_df: pd.DataFrame,
    val_df: pd.DataFrame,
    tokenizer: T5TokenizerFast,
    batch_size: int = 8,
    text_max_token_len: int = 512,
    summary_max_token_len: int = 128
  ):

    super ().__init__()

    self.train_df = train_df
    self.test_df = test_df
    self.val_df=val_df

    self.batch_size = batch_size
    self.tokenizer = tokenizer
```

```
        self.text_max_token_len = text_max_token_len
        self.summary_max_token_len = summary_max_token_len

    def setup(self, stage=None) :
        #print('test')
        self.train_dataset = SummaryDataset(
            self.train_df,
            self.tokenizer,
            self.text_max_token_len,
            self.summary_max_token_len
            )


        self.test_dataset = SummaryDataset(
            self.test_df,
            self.tokenizer,
            self.text_max_token_len,
            self.summary_max_token_len
            )


        self.val_dataset = SummaryDataset(
            self.val_df,
            self.tokenizer,
            self.text_max_token_len,
            self.summary_max_token_len
            )


    def train_dataloader(self):
        return DataLoader(
            self.train_dataset,
            batch_size=self.batch_size,
            shuffle=True,
            num_workers=2
            )

    def val_dataloader(self):
       return DataLoader(
            self.val_dataset,
            batch_size=self.batch_size,
            shuffle=False,
            num_workers=2
            )
    def test_dataloader(self):
       return DataLoader(
            self.test_dataset,
            batch_size=self.batch_size,
```

```
        shuffle=False,
        num_workers=2
        )

#Initialising tokenizer
modelName="t5-small"

tokenizer = T5TokenizerFast.from_pretrained(modelName)

text_token_counts =[]
summary_token_counts = []
#checking distribution of tokens in colums Text and Summary to get feeling
for _,row in train_data.iterrows():
  text_token_count = len(tokenizer.encode(row["Text"][:512]))
  text_token_counts.append(text_token_count)

  summary_token_count = len(tokenizer.encode(row["Summary"]))
  summary_token_counts.append(summary_token_count)

#Plotting lenght of text and summaries to see how many tokens we have each
fig, (ax1, ax2) = plt.subplots(1, 2)

sns.histplot(text_token_counts, ax=ax1)
ax1.set_title('full text token counts')

sns.histplot(summary_token_counts, ax=ax2)
ax2.set_title('summary text token counts')

!pip install datasets==1.0. #WE DONT NEED THIS OR?
!pip install rouge_score

import datasets
rouge=datasets.load_metric("rouge")
'''
def compute_metrics(pred):
    labels_ids=pred.label_ids
    pred_ids=pred.predictions

    # all unnecessary tokens are removed
    pred_str=tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    labels_ids[labels_ids==-100]=tokenizer.pad_token_id
    label_str=tokenizer.batch_decode(labels_ids, skip_special_tokens=True)
    print('pred_str'+str(pred_str))
    print('label_str'+str(label_str))
    rouge_output=rouge.compute(predictions=pred_str, references=label_str,

    return {
        "rouge2_precision": round(rouge_output.precision,4),
```

```
        "rouge2_recall": round(rouge_output.recall,4),
        "rouge2_fmeasure": round(rouge_output.fmeasure,4),
    }
'''

#Training parameters set up
N_EPOCHS = 3 #try more epochs, eg. 10 <-- whether it decreases, shows plate

TRAIN_BATCH_SIZE = 8 #Changing this from 16

BATCH_SIZE=16

data_module=SummaryDataModule(train_data,test_data,val_data,tokenizer,batch

"""### Model

"""

class SummaryModel(pl.LightningModule):

 def __init__(self):
   super().__init__()
   self.model = T5ForConditionalGeneration.from_pretrained(modelName, retur

 def forward(self,input_ids, attention_mask, decoder_attention_mask, labels
   output = self.model(
      input_ids,
      attention_mask=attention_mask,
      labels=labels,
      decoder_attention_mask=decoder_attention_mask
    )
   return output.loss, output.logits

 def training_step(self, batch, batch_idx):
   input_ids = batch[ "text_input_ids"]
   attention_mask = batch["text_attention_mask"]
   labels = batch["labels"]
   labels_attention_mask = batch["labels_attention_mask"]

   loss, outputs = self(
     input_ids=input_ids,
     attention_mask=attention_mask,
     decoder_attention_mask=labels_attention_mask,
     labels=labels
    )
   self.log("train loss", loss, prog_bar=True,logger=True)
   return loss

```

```
def validation_step(self, batch, batch_idx):
  input_ids = batch[ "text_input_ids"]
  attention_mask = batch["text_attention_mask"]
  labels = batch["labels"]
  labels_attention_mask = batch["labels_attention_mask"]

  loss, outputs = self(
    input_ids=input_ids,
    attention_mask=attention_mask,
    decoder_attention_mask=labels_attention_mask,
    labels=labels
  )
  self.log("val_loss", loss, prog_bar=True,logger=True)
  return loss

def compute_metrics(pred):
   labels_ids=pred.label_ids
   pred_ids=pred.predictions

   # all unnecessary tokens are removed
   pred_str=tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
   labels_ids[labels_ids==-100]=tokenizer.pad_token_id
   label_str=tokenizer.batch_decode(labels_ids, skip_special_tokens=True)
   print('pred_str'+str(pred_str))
   print('label_str'+str(label_str))
   rouge_output=rouge.compute(predictions=pred_str, references=label_str,

   return {
       "rouge2_precision": round(rouge_output.precision,4),
       "rouge2_recall": round(rouge_output.recall,4),
       "rouge2_fmeasure": round(rouge_output.fmeasure,4),
   }


def test_step(self, batch, batch_idx):
  input_ids = batch[ "text_input_ids"]
  attention_mask = batch["text_attention_mask"]
  labels = batch["labels"]
  labels_attention_mask = batch["labels_attention_mask"]

  loss, outputs = self(
    attention_mask=attention_mask,
    decoder_attention_mask=labels_attention_mask,
    labels=labels
   )
  self.log("test_loss", loss, prog_bar=True,logger=True)
  return loss

```

```
 def configure_optimizers(self):
    return AdamW(self.parameters(), lr=0.0001) #early_stopping: parameter t

model=SummaryModel()

# Commented out IPython magic to ensure Python compatibility.
# %load_ext tensorboard
# %tensorboard --logdir ./lightning_logs

checkpoint_callback = ModelCheckpoint(
  dirpath="checkpoints",
  filename="best-checkpoint",
  save_top_k=1,
  verbose=True,
  monitor="val_los",
  mode="min"
)


logger = TensorBoardLogger("lightning_logs", name="our-summary")

trainer = pl.Trainer(
  logger=logger,
  enable_checkpointing=checkpoint_callback,
  compute_metrics=compute_metrics,
  max_epochs=N_EPOCHS,
  gpus=1,
  enable_progress_bar = True
)

trainer.fit(model,data_module)

trained_model = SummaryModel.load_from_checkpoint(
  trainer.checkpoint_callback.best_model_path
)

trained_model.freeze()

def summarize (text):
  text_encoding = tokenizer(
    text,
    max_length=512,
    padding="max_length",
    truncation=True,
    return_attention_mask=True,
    add_special_tokens=True,
    return_tensors="pt"
  )
```

```
  generated_ids = trained_model.model.generate(
    input_ids=text_encoding["input_ids"],
    attention_mask=text_encoding["attention_mask"],
    max_length=200,
    num_beams=2,
    repetition_penalty=2.5,
    length_penalty=1.0,
    early_stopping=True
  )

  preds =[
    tokenizer.decode (gen_id, skip_special_tokens=True, clean_up_tokenizati
    for gen_id in generated_ids
  ]

  return "". join(preds)

test_data = test_data.reset_index()
for i in range(0,len(test_data)):
  test_data['Generated_summary'] = ""
test_data.head()

for i in range (len(test_data)):
  sample_row = test_data.iloc[i]
  text = sample_row["Text"]
  model_summary = summarize(text)
  test_data["Generated_summary"][i] = model_summary

test_data.head()

test_data.to_csv("/content/drive/MyDrive/summary_test.csv")

sample_row = test_data.iloc[6]
text = sample_row["Text" ]
model_summary = summarize(text)

#text

#summary = sample_row["Summary"]
#summary

#model_summary

#Calculate and print out rouge scores
scorer = rouge_scorer.RougeScorer(['rouge1', 'rougeL'], use_stemmer=True)
scores = scorer.score(text,model_summary)
scores
```

```
from datasets import load_metric
metric = load_metric("rouge")

def calc_rouge_scores(candidates, references):
    result = metric.compute(predictions=candidates, references=references,
    result = {key: round(value.mid.fmeasure * 100, 1) for key, value in res
    return result

import re
ref_summaries = list(test_data['Summary'])

for i in range (len(test_data)):
  candidate_summaries = list(test_data['Generated_summary'])
  print(f"First {i+1} sentence(s): Scores {calc_rouge_scores(candidate_summ

df.head()

print(len(df))

data = df.dropna(subset=['Generated_summary'])

print(len(data))

print(df.iloc[50])
```

A.2.2 Classification

```
# -*- coding: utf-8 -*-
"""Classification.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/176vFOTXiYvnVBSEzs6Yuii_GOHQ9hh
"""

#Preinstalling the necessary libraries
#Certain versions are required to avoid compatibility issues

from google.colab import drive
drive.mount('/content/drive')

!pip install numpy==1.19.5
!pip install tensorflow==2.7.0
!pip install transformers==4.7.0
!pip install sacremoses==0.0.45

#Importing necessary classes for classification and summarizaton
import tensorflow as tf
import tensorflow_datasets as tfds
from transformers import DistilBertTokenizerFast
from transformers import TFDistilBertForSequenceClassification

import pandas as pd
import numpy as np
import nltk
import re

nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer

from six import viewitems
#Importing methods for splitting and shuffling data (as dataset contains no
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit

#Katja PATH
df=pd.read_csv("/content/drive/MyDrive/DS Practical/Reviews.csv", engine="p

#Valari PATH
#df=pd.read_csv("/content/drive/MyDrive/Reviews.csv", engine="python", erro
df.drop(columns=['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNu
```

```
print("Before",len(df))
df = df.dropna()
print("Data size:",len(df))
df.head()

#Checking the avaliable GPUs (not necessary, made as a test of the system)

#num_gpus_available = len(tf.config.experimental.list_physical_devices('GPU
#print("Num GPUs Available: ", num_gpus_available)
#assert num_gpus_available > 0

'''
#Setting the dataset as a frame (trasforming it from tensor)
df = tfds.as_dataframe(df)
#Preview of the data
df.head()
'''

#Classifying the data into two classes: positive and negative based on thei
df["Sentiment"] = df["Score"].apply(lambda score: "positive" if score >= 3
df['Sentiment'] = df['Sentiment'].map({'positive':1, 'negative':0})

#df['short_review'] = df['Text'].str.decode("utf-8")

df = df[["Text", "Sentiment"]]

'''
#Dropping last n rows using drop
n = 54975
df.drop(df.tail(n).index,
        inplace = True)
'''

df=df.loc[1:10000]

df.dropna()
print("Data size:",len(df))

df.head()

#To check how big is the dataset / num of rows
#index = df.index
#number_of_rows = len(index)
#print(number_of_rows)

#Printing the beginning part to see if the data is read correctly
#df.head()

```

```
#Printing the beginning part to see if the data is read correctly
#df.tail()

#Testing the labels
reviews = df['Text'].values.tolist()
labels = df['Sentiment'].tolist() #convert to category
#print(reviews[:2])
#print(labels[:2])

#training_sentences, validation_sentences, training_labels, validation_labe
training_sentences, validation_sentences, training_labels, validation_label
#this is on creating stratifyed sample

#Preprocessing the data using DistilBert for punctuation splitting and word
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncase

tokenizer([training_sentences[0]], truncation=True,
                            padding=True, max_length=128)

train_encodings = tokenizer(training_sentences,
                            truncation=True,
                            padding=True)
val_encodings = tokenizer(validation_sentences,
                            truncation=True,
                            padding=True)

train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    training_labels
))

val_dataset = tf.data.Dataset.from_tensor_slices((
    dict(val_encodings),
    validation_labels
))

print(val_dataset)

#tbd
model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-b

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5, epsilon=1e-08)
callbacks=tf.keras.callbacks.EarlyStopping(
    monitor='accuracy',
    min_delta=0.0001,
    patience=3,
    mode='auto',
    verbose=2,
```

```
    baseline=None
)
model.compile(optimizer=optimizer, loss=model.compute_loss, metrics=['accur
model.fit(train_dataset.shuffle(100).batch(16),
          epochs=2,
          batch_size=16,
          validation_data=val_dataset.shuffle(100).batch(16),callbacks=call

'''
import matplotlib.pyplot as plt

plt.title('Loss curves')
plt.plot(model.train_loss_history, '-', label='train')
plt.plot(model.val_loss_history, '-', label='val')
plt.legend(loc='lower right')
plt.xlabel('Iteration')
plt.show()

'''

model.save_pretrained("./sentiment")

loaded_model = TFDistilBertForSequenceClassification.from_pretrained("./ser

import pandas as pd
#Testing a model with a user-written input

#df = pd.DataFrame({'Text': ["This is a not a good product. I hate it", "Th
#test_sentence = "This is a not a good product. I hate it"

df1 = pd.read_csv("/content/drive/MyDrive/summary_test.csv", index_col=0, e
df1=df1.loc[0:50]
selected_columns = df1[["Summary","Text","Generated_summary"]]
df = selected_columns.copy()
df = df.dropna()


df.head()

for i in range(0, len(df)):
  df['Sentiment_text'] = i
  df['Sentiment_summary'] = i

for i in range(0,len(df)):

  predict_input_text = tokenizer.encode(df['Text'][i],
                                 truncation=True,
                                 padding=True,
```

```
                                 return_tensors="tf")
  tf_output_text = loaded_model.predict(predict_input_text)[0]
  tf_prediction_text = tf.nn.softmax(tf_output_text, axis=1)
  labels = ['Negative','Positive']
  label_text = tf.argmax(tf_prediction_text, axis=1)
  label_text = label_text.numpy()
  df["Sentiment_text"][i] = (labels[label_text[0]])


#df = df.append(data, columns = "Sentiment")
#print(df['Text'], df['Sentiment_text'])

for i in range(0, len(df)):

  predict_input_sum = tokenizer.encode(df['Generated_summary'][i],
                                 truncation=True,
                                 padding=True,
                                 return_tensors="tf")
  tf_output_sum = loaded_model.predict(predict_input_sum)[0]
  tf_prediction_sum = tf.nn.softmax(tf_output_sum, axis=1)
  #labels = ['Negative','Positive']
  label_sum = tf.argmax(tf_prediction_sum, axis=1)
  label_sum = label_sum.numpy()
  df["Sentiment_summary"][i] = (labels[label_sum[0]])


#print(df['Generated_summary'], df['Sentiment_summary'])
df.head()

tag = 0
for i in range(0, len(df)):
  if (df['Sentiment_text'][i] != df['Sentiment_summary'][i]):
    tag = tag + 1

Error = tag/len(df)
print(Error)
```

A.3 Scores outputs (graphical)