

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 25th, 2022



Supervised by Dr Maka Karalashvili and Dr Matthias Schubert

Before starting a vehicle model series production, the production process of it is tested within vehicle concept and prototype engineering. Data is collected, specifically, to track occurring quality defects needing the rework. For ease of presentation, this data will be referred to as “Prototype”.

When a vehicle model goes into series production in a plant, during production, again, very similar data is collected to record each quality defect that again needs to go into the rework. This data will be referred to as “Production”. For defects occurring during series production respective tickets are raised. All these tickets should be resolved by the end of vehicle production.

Quality defect recordings in either of the mentioned data sources exhibit a human, free text description. To better maintain these defects a more manageable, superordinate data source – referred to as “Knowledge-Base” – is built with the purpose 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.

Contents

1	Related work	1
2	Theoretical aspects	3
2.1	The task of summarization	3
2.2	The task of classification	3
2.3	Existing methods and approaches	3
2.4	Model choice	6
2.4.1	Summarization	6
2.4.2	Classification	7
3	Background prerequisites	8
3.1	Project goals	8
3.2	Data cleansing	8
3.3	Data processing	9
4	Implementation storyline	11
4.1	Project storyline	11
4.2	Implementation aspects	13
4.3	Performance analysis	13
5	Future work	15
6	Conclusion	16
7	References	17
A	Appendix	V
A.1	The list of the Python packages, used in the project	V
A.2	Implementation code	VII
A.2.1	Summarization	VII
A.2.2	Classification	XIX
A.3	Set up	XXIII
A.4	Scores outputs	XXIII

1 Related work

Natural Language Processing (NLP) is a machine learning field, that allows computer to analyze, control, and possibly create human dialect. Over the last years, it is getting more and 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 on eleven different NLP tasks (f. eg. next sentence prediction, question answering or generating the summary of the written text). Different BERT models were trained with various approaches, such as diversing the number of layers, hidden units, and attention heads, using the same hyperparameters and training procedure. These test 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, that makes it very compute-intensive at inference time. As an answer to such limitation, many new models for optimizing the BERT were generated, for example, RoBERTa (a robustly optimized BERT approach) ([2], [3]). The main differences 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 in 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 task as summarisation. Singh [5] suggests using it to create an abstractive summary for 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], [noauthor’ summarisation’ nodate]). 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. Despite GPT-2 showing performance similar or lesser than classic models that were trained for summarisation [8], GPT-3 GPT-3’s summarisation functionality is a great way to create summaries for books, papers, and articles [noauthor’ summarisation’ nodate].

Lately, a transfer learning has proven to be a powerful technique in NLP. The idea behind it is to first pre-train the model on a data-rich task 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) ([9], [10]). T5 model has 11 billion parameters and showed a great performance on 17 NLP tasks, for example, text classification [11], data to text generation [12] and summarisation [13]. 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 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

only 512 tokens of the input.

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

2 Theoretical aspects

2.1 The task of summarization

According to [14], a **summary** can be defined as a text that is produced from one or more texts, that contains 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 long text processing 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:

1. **Extractive:** Here, important text or sentences are extracted as they appear in the original document and are grouped to form a concise summary. Most extractive summarization techniques focus on finding and extracting Keywords from the parent text. One can say that it is similar to highlighting the most important parts of the text with the marker.
2. **Abstractive:** This approach focuses on generating summaries using the important ideas or facts, that are contained in the document without repeating them verbatim. It is similar to a summary that a person would write after reading the text.

2.2 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:

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 are an indication of 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.

2.3 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 very easy to use, as an algorithm needs to be imported without the necessity of much coding or fine-tuning. However, most of the algorithms in Sumy are supposed to be used for extractive summarization.

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.

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.
 - **Classification:** The Gensim library provides the Doc2vec algorithm which 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 [15].
- *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 [16]. For NLP, the most widely used Deep learning algorithms are Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). CNN are traditionally used in computer vision tasks but recent research has shown that they are as well very effective on NLP tasks. RNN are specifically designed to process sequential information.
 - **Summarization:** CNNs have mostly been implemented to only perform extractive text summarization. However, these models have a very complex architecture, are computationally very expensive, and hard to interpret. It is also very hard to implement deep bidirectionality using CNNs.

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.

- **Classification:** CNNs can be used for classification by utilizing a feature that is applied to words or n-grams to extract high-level features [16].

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.

- *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) [17], 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) [18], etc.

Since BERT and Co can be easily fine-tuned to perform any NLP tasks, 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 returns a modified text string as the output. This gives the T5 model an advantage over BERT-based models as the latter only return 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 [13].

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 [19]. 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 [20]. Though, when it comes to summarization, it is outperformed by T5 [6].

2.4 Model choice

Over the past few years, transfer learning has led to a new wave of the 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 modeling or filling in the missing words. After that, the model can be fine-tuned on smaller labeled datasets, often resulting in a better performance than training on the labeled 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 difficult to evaluate which improvements are most meaningful and how effective they are when combined.

2.4.1 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, authors decided to confine to "T5-small" version, that was as well pre-trained on a multi-task mixture of unsupervised and supervised tasks, and performs not worse, than its extended variations [21].

2.4.2 Classification

Due to the benefits of transfer learning, 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 very good choice. The developers of this project were confined to limited computational resources, it was decided to implement the DistilBERT model. DistilBERT is much smaller, faster and cheaper model, compared to BERT and has provided SOTA results for classification tasks [18].

Implementation aspects and computational results are as well provided in further chapters of this work (c.f. Sec. 4.2 for the details on implementation and Sec. 4.3 for the performance analysis).

3 Background prerequisites

3.1 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.

The quality metrics can have two target audiences: the first one is a *Data Steward*, a person not necessarily equipped with a Data Science background, and the second one is a *Data Scientist*. This implies that it should be easily understood by persons without a Data Science background. Moreover, the choice of the metrics, that are being used in the project, strongly depends on the available data, which extends the tasks of the metric derivation and output of the expected results.

Due to several data security issues, analyzing and handling the initial data was impossible. Hence, the research for the most similar structured data 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 existent resources. Thereby, authors had narrow 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.

3.2 Data cleansing

Amazon Product Review dataset is a publicly available dataset [22], which contains 568454 reviews. It contains 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 568427 reviews.

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

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 the computational resources which were at the disposal, authors could not perform the training task on the whole dataset. Hence, a larger subsample of around 30 000 reviews, was used.

Similarly, for classification tasks, author 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.

3.3 Data processing

Both summarization and classification tasks can be performed with various models that are implemented 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 Google T5 model was used, and the DistilBERT model as a classification, both already implemented in the Transformers package. To be able to train 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. 2. As a consequence of available computation power, we have implemented a t5-small model for generating summaries of Reviews.

<i>Model size variants</i>						
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 2: T5 model size variants [10]

For the T5-small model, we have split our dataset into train, validation, and test datasets in the ratio of 80-10-10. Afterward, each dataset was encoded using **T5TokenizerFast** from the Transformers package to suit 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 about this is illustrated in A.2.

Similarly was implemented for the Classification model. Instead of the T5 model here was used the DistilBERT model, implemented in the Transformers package as well. The Text field is used as input and *Sentiment* as a label to train the model. Different from the split of the dataset for the T5 summarization model, here was used stratified split by 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. We then proceed to fit a pre-trained DistilBert Classification model on our prepared dataset.

4 Implementation storyline

4.1 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, dataset to use and suggested to start the research of the possible methods. However, as stated in the 3.1, due to various issues with the access to the data and data security, there arose the necessity to conduct an additional research of the plausible for the task open-source data (Fig. 3).

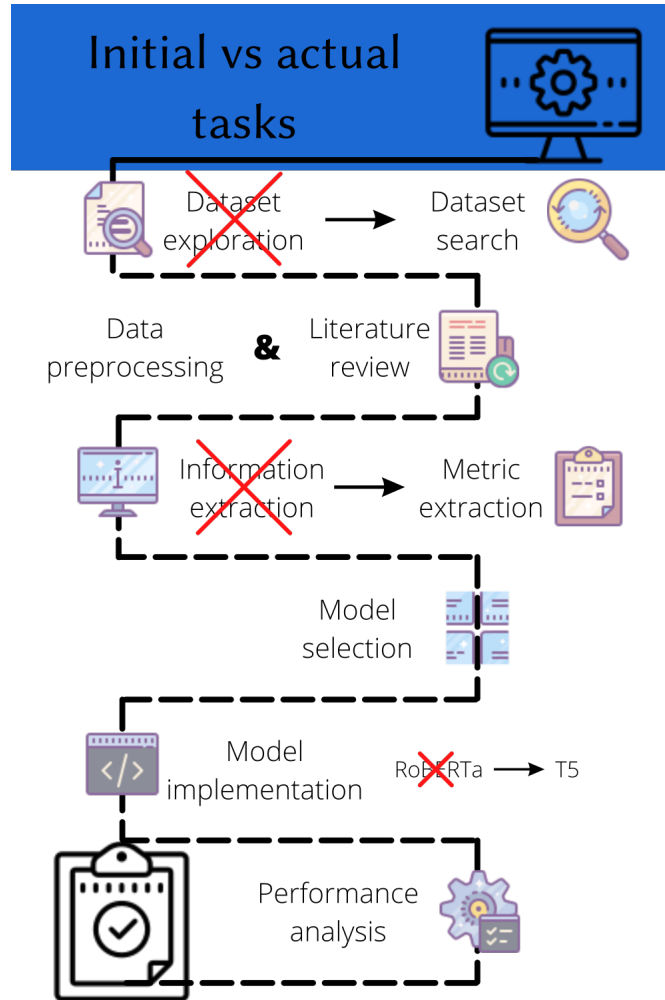


Figure 3: The scope of the project, including tasks, changed due to unexpected issues

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

Metric selection and extraction As a result of the research, the Amazon Product Reviews datasets [22] 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, mood of the review, product names within it.
3. **overall** score, given by user, should be consistent with the mood of the review itself.

However, during the approach discussion, it was decided to reduce 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 choice of the models to use was described more precisely in the Sec. 2.4. Moreover, together with the model selection, a bigger and more applicable dataset was found for the better analysis. The new dataset was as well containing the Amazon Product reviews, yet still, a different preprocessing was needed (c.f. Sec. 3.2).

At the same time, literature research and review were conducted (c.f. Sec. 1). 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. 4.2 of the current report. The first model, that was selected for implementation was RoBERTa [4]. As mentioned in Sec. 2.4.1, it is an encoder model, that suffers on some issues with weight assignment to the decoder. Thus, although the model was correctly implemented and adapted to chosen data, the training of it was performed incorrectly. Hence, T5 model was chosen as it is more robust and easier in issue handling.

Performance analysis All outputs and their interpretation is thoroughly discussed in the Sec. 4.3.

Needless to say, that every complicated model requires a lot of work and fixing. Hence, the authors of this report came across several issues during the project, that affected the deadlines and resulted in prolongation of the project (c.f. Fig. 4).

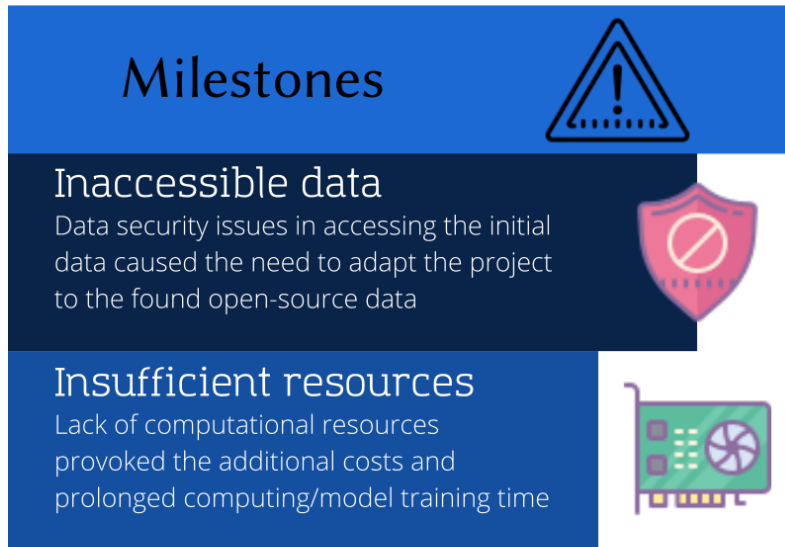


Figure 4: The most troublesome aspects of the project, that resulted in the necessity to expand the deadlines

4.2 Implementation aspects

Section 2.1 of the current document refers to the different types of summarizations. To be able to test both summarization methods, the standard BERT library (Bidirectional Encoder Representations from Transformers [2]) was chosen. Although, for easier and faster computation the DistilBERT (distilled version of BERT [3]) is being used. Additionally, a Robustly optimised BERT approach (RoBERTa) was applied and tested for creating both abstractive and extractive summaries.

After creating the summary, it should be classified. Another model was created for that part, using BERT and NLTK libraries which are currently working with over 90% accuracy.

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, thus, it could not learn anything during training. Further research showed that Text-to-Text Transfer Transformer (T5), despite being bulkier than the previously tested RoBERTa model, is more accurate in performing summarization tasks and its tokenizer is more robust. Moreover, the T5 Model suggests easier issue handling.

4.3 Performance analysis

Performance metrics:

- 1.

Computation time

5 Future work

6 Conclusion

A concise summary of contents and results

7 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](https://doi.org/10.48550/arXiv.1810.04805). arXiv: [1810.04805\[cs\]](https://arxiv.org/abs/1810.04805). 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\[cs\]](https://arxiv.org/abs/1907.11692). 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] 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).
- [9] 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).
- [10] 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).
- [11] 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).
- [12] 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).

- [13] *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).
- [14] Eduard Hovy. *Text Summarization*. The Oxford Handbook of Computational Linguistics. ISBN: 9780199276349. Jan. 13, 2005. DOI: [10.1093/oxfordhb/9780199276349.013.0032](https://doi.org/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).
- [15] 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).
- [16] 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).
- [17] 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](https://doi.org/10.48550/arXiv.1909.11942). arXiv: [1909.11942\[cs\]](https://arxiv.org/abs/1909.11942). URL: <http://arxiv.org/abs/1909.11942> (visited on 05/22/2022).
- [18] 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](https://doi.org/10.48550/arXiv.1910.01108). arXiv: [1910.01108\[cs\]](https://arxiv.org/abs/1910.01108). URL: <http://arxiv.org/abs/1910.01108> (visited on 05/22/2022).
- [19] *Summarization with GPT-3*. KDnuggets. Section: Products and Services. URL: <https://www.kdnuggets.com/summarization-with-gpt-3.html/> (visited on 05/22/2022).
- [20] 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).
- [21] Zhanpeng Wang. “Smart Auto-completion in Live Chat Utilizing the Power of T5”. PhD thesis.
- [22] *Amazon review data*. URL: <https://jmcauley.ucsd.edu/data/amazon/> (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	
DistilBertTokenizerFast	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: StratifiedShuffleSplit	
T5ForConditionalGeneration	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	
TFDistilBertForSequenceClassification	classification	HuggingFace: DistilBert	

torch	summarization	Pytorch	
train_test_split	classification	sklearn: train_test_split	
Trainer	summarization	Transformers: Trainer	
TrainingArguments	summarization	Transformers: TrainingArguments	
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

```

1  # -*- coding: utf-8 -*-
2  """T5Summarization.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/1kYVQqol5iIwEye1nm5cXZ27IkIMk1K
8  """
9
10 #Preinstalling necessary libraries / speciflicated versions are necessary to
11
12 !pip install --quiet transformers==4.5
13
14 !pip install --quiet rouge_score
15
16 !pip install --quiet pytorch-lightning
17
18 !pip install --quiet tensorflow
19 !pip install --quiet tensorboard
20 !pip install --quiet nltk
21
22 #importing necessary libraries and packages
23 import json
24 import pandas as pd
25 import numpy as np
26 import logging
27 import torch
28 from torch.utils.data import DataLoader, Dataset
29 import pytorch_lightning as pl
30 import matplotlib.pyplot as plt
31 import seaborn as sns
32
33
34 from dataclasses import dataclass, field
35 from typing import Optional
36 #importing tokenizer and T5 model
37 from transformers import T5ForConditionalGeneration, T5TokenizerFast, AdamW
38
39
40 from pytorch_lightning.callbacks import ModelCheckpoint

```



```

41 from pytorch_lightning.loggers import TensorBoardLogger
42 from rouge_score import rouge_scorer
43
44 #packages and libraries for removing stopwords
45 import re
46 import nltk
47 nltk.download('stopwords')
48 nltk.download('punkt')
49 from nltk.corpus import stopwords
50 from nltk.tokenize import word_tokenize
51
52 #Importing Google Drive for reading the data
53 from google.colab import drive
54 drive.mount('/content/drive')
55
56 #Reading and clearing the data
57 #Vladana PATH
58 df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Reviews.csv", engine=
59
60
61 #Vallari PATH
62 #df=pd.read_csv("/content/drive/MyDrive/Reviews.csv", engine="python", erro
63
64 #Katja PATH
65 #df=pd.read_csv("/content/drive/MyDrive/DS Practical/Reviews.csv", engine="
66
67 df.drop(columns=['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNu
68 #print("Before",len(df))
69 df = df.dropna()
70 #print("Data size:",len(df))
71 df.head()
72
73 #Shortening the data for testing purposes (remove the whole cell for full t
74 df1=df.loc[1:2000]
75 #df.shape
76 #df1.shape
77 print("Data size:",len(df1))
78
79 tokens_wo_stopwords[:512]
80
81 #Untokenize function from
82 #https://github.com/commonsense/metanl/blob/master/metanl/token_utils.py
83 def untokenize(words):
84     """
85     Untokenizing a text undoes the tokenizing operation, restoring
86     punctuation and spaces to the places that people expect them to be.
87     Ideally, `untokenize(tokenize(text))` should be identical to `text`,
88     except for line breaks.

```

```

89     """
90     text = ' '.join(words)
91     step1 = text.replace("` `", "'').replace(" '", "'').replace('. . .',
'|...')
92     step2 = step1.replace(" ( ", " (").replace(" ) ", ") ")
93     step3 = re.sub(r' ([.,:;?!%]+)([ \'"`])', r"\1\2", step2)
94     step4 = re.sub(r' ([.,:;?!%]+)$', r"\1", step3)
95     step5 = step4.replace(" '", "'').replace(" n't", "n't").replace(
96         "can not", "cannot")
97     step6 = step5.replace(" ` ", " ' ")
98     return step6.strip()
99
100 text = df1['Text']
101
102 #Convert text to lowercase and split to a list of words
103 tokens=[]
104 for i in range(len(text)):
105     oneRow=text.iloc[i]
106     tokens.append(word_tokenize(oneRow.lower()))
107
108
109
110 #Remove stop words
111 english_stopwords = stopwords.words('english')
112 tokensWoStopwords=[]
113
114 for i in range(len(tokens)):
115     tokens_wo_stopwords = [t for t in tokens[i] if t not in english_stopwords]
116     #print(len(tokens_wo_stopwords))
117     tokensWoStopwords.append(untokenize(tokens_wo_stopwords[:512]))
118
119 #print(len(tokens_wo_stopwords))
120
121 len(i.split(' '))
122
123 i
124
125 for i in tokensWoStopwords:
126     if len(i.split(' '))>512:
127         print(i)
128         print('-----')
129
130 #replacing Text with Text without stopwords
131 df1['Text']=tokensWoStopwords
132 df1=df1.reset_index(drop=True)
133 df1.head
134
135 #Shortened dataset split into train , validation and test dataset

```

```

136 n_train = int(np.round(df1.shape[0]*0.8))
137 n_val = int(np.round(df1.shape[0]*0.1))
138 n_test = int(np.round(df1.shape[0]*0.1))
139 train_data=df1.loc[:n_train]
140 val_data=df1.loc[n_train:n_train+n_val]
141 test_data=df1.loc[n_train+n_val:n_train+n_val+n_test]
142
143
144 #Full dataset split
145 #n_train = int(np.round(df.shape[0]*0.8))
146 #n_val = int(np.round(df.shape[0]*0.1))
147 #n_test = int(np.round(df.shape[0]*0.1))
148 #train_data=df.loc[:n_train]
149 #val_data=df.loc[n_train:n_train+n_val]
150 #test_data=df.loc[n_train+n_val:n_train+n_val+n_test]
151
152 #Checking how dataset is splitted
153 #train_data.shape, test_data.shape, val_data.shape
154
155 #Creating dataset shape for new T5 model for summarisation
156 class SummaryDataset (Dataset):
157     def __init__ (
158         self,
159         data: pd.DataFrame,
160         tokenizer: T5TokenizerFast, #initializing tokenizer
161         text_max_token_len: int = 512, #setting maximum lenght of tokens for
162         summary_max_token_len: int = 128 #setting maximum lenght of tokens fo
163     ):
164         self.tokenizer = tokenizer
165         self.dataF = data
166         self.text_max_token_len = text_max_token_len
167         self.summary_max_token_len = summary_max_token_len
168
169     def __len__(self):
170         return len(self.dataF)
171
172     def __getitem__(self, index: int):
173         data_row = self.dataF.iloc[index]
174
175         text = data_row["Text"]
176         #encoding Text value to be suitable for pretrained T5 model
177         text_encoding = tokenizer(
178             text,
179             max_length=self.text_max_token_len,
180             padding="max_length",
181             truncation=True,
182             return_attention_mask=True,
183             add_special_tokens=True,

```

```

184         return_tensors="pt"
185
186     )
187     #encoding Summary value to be suitable for pretrained T5 model
188     summary_encoding = tokenizer(
189         data_row["Summary"],
190         max_length=self.summary_max_token_len,
191         padding="max_length",
192         truncation=True,
193         return_attention_mask=True,
194         add_special_tokens=True,
195         return_tensors="pt"
196
197     )
198
199     labels = summary_encoding["input_ids"]
200     labels[labels==0]=-100
201
202     return dict(
203         text=text,
204         summary=data_row["Summary"],
205         text_input_ids=text_encoding["input_ids"].flatten(),
206         text_attention_mask=text_encoding["attention_mask"].flatten(),
207         labels=labels.flatten(),
208         labels_attention_mask=summary_encoding["attention_mask"].flatten()
209     )
210
211 # encoding train, validation and test dataset to desired input of model
212 class SummaryDataModule(pl.LightningDataModule):
213     def __init__(
214         self,
215         train_df: pd.DataFrame,
216         test_df: pd.DataFrame,
217         val_df: pd.DataFrame,
218         tokenizer: T5TokenizerFast,
219         batch_size: int = 8,
220         text_max_token_len: int = 512,
221         summary_max_token_len: int = 128
222     ):
223
224         super().__init__()
225
226         self.train_df = train_df
227         self.test_df = test_df
228         self.val_df=val_df
229
230         self.batch_size = batch_size
231         self.tokenizer = tokenizer

```

```
232     self.text_max_token_len = text_max_token_len
233     self.summary_max_token_len = summary_max_token_len
234
235     def setup(self, stage=None) :
236         #print('test')
237         self.train_dataset = SummaryDataset(
238             self.train_df,
239             self.tokenizer,
240             self.text_max_token_len,
241             self.summary_max_token_len
242         )
243
244
245         self.test_dataset = SummaryDataset(
246             self.test_df,
247             self.tokenizer,
248             self.text_max_token_len,
249             self.summary_max_token_len
250         )
251
252
253         self.val_dataset = SummaryDataset(
254             self.val_df,
255             self.tokenizer,
256             self.text_max_token_len,
257             self.summary_max_token_len
258         )
259
260
261     def train_dataloader(self):
262         return DataLoader(
263             self.train_dataset,
264             batch_size=self.batch_size,
265             shuffle=True,
266             num_workers=2
267         )
268
269     def val_dataloader(self):
270         return DataLoader(
271             self.val_dataset,
272             batch_size=self.batch_size,
273             shuffle=False,
274             num_workers=2
275         )
276     def test_dataloader(self):
277         return DataLoader(
278             self.test_dataset,
279             batch_size=self.batch_size,
```

```

280         shuffle=False,
281         num_workers=2
282     )
283
284 #Initialising tokenizer
285 modelName="t5-small"
286
287 tokenizer = T5TokenizerFast.from_pretrained(modelName)
288
289 text_token_counts = []
290 summary_token_counts = []
291 #checking distribution of tokens in columns Text and Summary to get feeling
292 for _,row in train_data.iterrows():
293     text_token_count = len(tokenizer.encode(row["Text"][:512]))
294     text_token_counts.append(text_token_count)
295
296     summary_token_count = len(tokenizer.encode(row["Summary"]))
297     summary_token_counts.append(summary_token_count)
298
299 #Plotting length of text and summaries to see how many tokens we have each
300 fig, (ax1, ax2) = plt.subplots(1, 2)
301
302 sns.histplot(text_token_counts, ax=ax1)
303 ax1.set_title('full text token counts')
304
305 sns.histplot(summary_token_counts, ax=ax2)
306 ax2.set_title('summary text token counts')
307
308 !pip install datasets==1.0. #WE DONT NEED THIS OR?
309 !pip install rouge_score
310
311 import datasets
312 rouge=datasets.load_metric("rouge")
313 '''
314 def compute_metrics(pred):
315     labels_ids=pred.label_ids
316     pred_ids=pred.predictions
317
318     # all unnecessary tokens are removed
319     pred_str=tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
320     labels_ids[labels_ids==-100]=tokenizer.pad_token_id
321     label_str=tokenizer.batch_decode(labels_ids, skip_special_tokens=True)
322     print('pred_str'+str(pred_str))
323     print('label_str'+str(label_str))
324     rouge_output=rouge.compute(predictions=pred_str, references=label_str,
325
326     return {
327         "rouge2_precision": round(rouge_output.precision,4),

```

```

328         "rouge2_recall": round(rouge_output.recall,4),
329         "rouge2_fmeasure": round(rouge_output.fmeasure,4),
330     }
331 '''
332
333 #Training parameters set up
334 N_EPOCHS = 3 #try more epochs, eg. 10 <-- whether it decreases, shows plate
335
336 TRAIN_BATCH_SIZE = 8 #Changing this from 16
337
338 BATCH_SIZE=16
339
340 data_module=SummaryDataModule(train_data,test_data,val_data,tokenizer,batch
341
342 """### Model
343
344 """
345
346 class SummaryModel(pl.LightningModule):
347
348     def __init__(self):
349         super().__init__()
350         self.model = T5ForConditionalGeneration.from_pretrained(modelName, retur
351
352     def forward(self,input_ids, attention_mask, decoder_attention_mask, labels
353         output = self.model(
354             input_ids,
355             attention_mask=attention_mask,
356             labels=labels,
357             decoder_attention_mask=decoder_attention_mask
358         )
359         return output.loss, output.logits
360
361     def training_step(self, batch, batch_idx):
362         input_ids = batch["text_input_ids"]
363         attention_mask = batch["text_attention_mask"]
364         labels = batch["labels"]
365         labels_attention_mask = batch["labels_attention_mask"]
366
367         loss, outputs = self(
368             input_ids=input_ids,
369             attention_mask=attention_mask,
370             decoder_attention_mask=labels_attention_mask,
371             labels=labels
372         )
373         self.log("train loss", loss, prog_bar=True,logger=True)
374         return loss
375

```

```

376 def validation_step(self, batch, batch_idx):
377     input_ids = batch["text_input_ids"]
378     attention_mask = batch["text_attention_mask"]
379     labels = batch["labels"]
380     labels_attention_mask = batch["labels_attention_mask"]
381
382     loss, outputs = self(
383         input_ids=input_ids,
384         attention_mask=attention_mask,
385         decoder_attention_mask=labels_attention_mask,
386         labels=labels
387     )
388     self.log("val_loss", loss, prog_bar=True, logger=True)
389     return loss
390
391 def compute_metrics(pred):
392     labels_ids=pred.label_ids
393     pred_ids=pred.predictions
394
395     # all unnecessary tokens are removed
396     pred_str=tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
397     labels_ids[labels_ids==-100]=tokenizer.pad_token_id
398     label_str=tokenizer.batch_decode(labels_ids, skip_special_tokens=True)
399     print('pred_str'+str(pred_str))
400     print('label_str'+str(label_str))
401     rouge_output=rouge.compute(predictions=pred_str, references=label_str,
402
403     return {
404         "rouge2_precision": round(rouge_output.precision,4),
405         "rouge2_recall": round(rouge_output.recall,4),
406         "rouge2_fmeasure": round(rouge_output.fmeasure,4),
407     }
408
409
410 def test_step(self, batch, batch_idx):
411     input_ids = batch["text_input_ids"]
412     attention_mask = batch["text_attention_mask"]
413     labels = batch["labels"]
414     labels_attention_mask = batch["labels_attention_mask"]
415
416     loss, outputs = self(
417         attention_mask=attention_mask,
418         decoder_attention_mask=labels_attention_mask,
419         labels=labels
420     )
421     self.log("test_loss", loss, prog_bar=True, logger=True)
422     return loss
423

```



```

424     def configure_optimizers(self):
425         return AdamW(self.parameters(), lr=0.0001) #early_stopping: parameter t
426
427     model=SummaryModel()
428
429     # Commented out IPython magic to ensure Python compatibility.
430     # %load_ext tensorboard
431     # %tensorboard --logdir ./lightning_logs
432
433     checkpoint_callback = ModelCheckpoint(
434         dirpath="checkpoints",
435         filename="best-checkpoint",
436         save_top_k=1,
437         verbose=True,
438         monitor="val_loss",
439         mode="min"
440     )
441
442
443     logger = TensorBoardLogger("lightning_logs", name="our-summary")
444
445     trainer = pl.Trainer(
446         logger=logger,
447         enable_checkpointing=checkpoint_callback,
448         compute_metrics=compute_metrics,
449         max_epochs=N_EPOCHS,
450         gpus=1,
451         enable_progress_bar = True
452     )
453
454     trainer.fit(model,data_module)
455
456     trained_model = SummaryModel.load_from_checkpoint(
457         trainer.checkpoint_callback.best_model_path
458     )
459
460     trained_model.freeze()
461
462     def summarize (text):
463         text_encoding = tokenizer(
464             text,
465             max_length=512,
466             padding="max_length",
467             truncation=True,
468             return_attention_mask=True,
469             add_special_tokens=True,
470             return_tensors="pt"
471         )

```

```

472     generated_ids = trained_model.model.generate(
473         input_ids=text_encoding["input_ids"],
474         attention_mask=text_encoding["attention_mask"],
475         max_length=200,
476         num_beams=2,
477         repetition_penalty=2.5,
478         length_penalty=1.0,
479         early_stopping=True
480     )
481
482     preds = [
483         tokenizer.decode (gen_id, skip_special_tokens=True, clean_up_tokenization_spaces=True)
484         for gen_id in generated_ids
485     ]
486
487     return "".join(preds)
488
489 test_data = test_data.reset_index()
490 for i in range(0,len(test_data)):
491     test_data['Generated_summary'] = ""
492 test_data.head()
493
494 for i in range (len(test_data)):
495     sample_row = test_data.iloc[i]
496     text = sample_row["Text"]
497     model_summary = summarize(text)
498     test_data["Generated_summary"][i] = model_summary
499
500 test_data.head()
501
502 test_data.to_csv("/content/drive/MyDrive/summary_test.csv")
503
504 sample_row = test_data.iloc[6]
505 text = sample_row["Text"]
506 model_summary = summarize(text)
507
508 #text
509
510 #summary = sample_row["Summary"]
511 #summary
512
513 #model_summary
514
515 #Calculate and print out rouge scores
516 scorer = rouge_scorer.RougeScorer(['rouge1', 'rougeL'], use_stemmer=True)
517 scores = scorer.score(text,model_summary)
518 scores
519

```

```
520 from datasets import load_metric
521 metric = load_metric("rouge")
522
523 def calc_rouge_scores(candidates, references):
524     result = metric.compute(predictions=candidates, references=references,
525     result = {key: round(value.mid.fmeasure * 100, 1) for key, value in res
526     return result
527
528 import re
529 ref_summaries = list(test_data['Summary'])
530
531 for i in range(len(test_data)):
532     candidate_summaries = list(test_data['Generated_summary'])
533     print(f"First {i+1} sentence(s): Scores {calc_rouge_scores(candidate_summ
534
535 df.head()
536
537 print(len(df))
538
539 data = df.dropna(subset=['Generated_summary'])
540
541 print(len(data))
542
543 print(df.iloc[50])
```

A.2.2 Classification

```

1  # -*- coding: utf-8 -*-
2  """Classification.ipynb
3
4  Automatically generated by Colaboratory.
5
6  Original file is located at
7      https://colab.research.google.com/drive/176vF0TXiYvnVBSEzs6Yuii_G0HQ9hh
8  """
9
10 #Preinstalling the necessary libraries
11 #Certain versions are required to avoid compatibility issues
12
13 from google.colab import drive
14 drive.mount('/content/drive')
15
16 !pip install numpy==1.19.5
17 !pip install tensorflow==2.7.0
18 !pip install transformers==4.7.0
19 !pip install sacremoses==0.0.45
20
21 #Importing necessary classes for classification and summarization
22 import tensorflow as tf
23 import tensorflow_datasets as tfds
24 from transformers import DistilBertTokenizerFast
25 from transformers import TFDistilBertForSequenceClassification
26
27 import pandas as pd
28 import numpy as np
29 import nltk
30 import re
31
32 nltk.download('stopwords')
33 from nltk.corpus import stopwords
34 from nltk.stem.porter import PorterStemmer
35
36 from six import viewitems
37 #Importing methods for splitting and shuffling data (as dataset contains no
38 from sklearn.model_selection import train_test_split
39 from sklearn.model_selection import StratifiedShuffleSplit
40
41 #Katja PATH
42 df=pd.read_csv("/content/drive/MyDrive/DS Practical/Reviews.csv", engine="p
43
44 #Valari PATH
45 #df=pd.read_csv("/content/drive/MyDrive/Reviews.csv", engine="python", erro
46 df.drop(columns=['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNu

```

```

47 print("Before",len(df))
48 df = df.dropna()
49 print("Data size:",len(df))
50 df.head()
51
52 #Checking the available GPUs (not necessary, made as a test of the system)
53
54 #num_gpus_available = len(tf.config.experimental.list_physical_devices('GPU'))
55 #print("Num GPUs Available: ", num_gpus_available)
56 #assert num_gpus_available > 0
57
58 '''
59 #Setting the dataset as a frame (transforming it from tensor)
60 df = tfds.as_dataframe(df)
61 #Preview of the data
62 df.head()
63 '''
64
65 #Classifying the data into two classes: positive and negative based on their score
66 df["Sentiment"] = df["Score"].apply(lambda score: "positive" if score >= 3 else "negative")
67 df['Sentiment'] = df['Sentiment'].map({'positive':1, 'negative':0})
68
69 #df['short_review'] = df['Text'].str.decode("utf-8")
70
71 df = df[["Text", "Sentiment"]]
72
73 '''
74 #Dropping last n rows using drop
75 n = 54975
76 df.drop(df.tail(n).index,
77         inplace = True)
78 '''
79
80 df=df.loc[1:10000]
81
82 df.dropna()
83 print("Data size:",len(df))
84
85 df.head()
86
87 #To check how big is the dataset / num of rows
88 #index = df.index
89 #number_of_rows = len(index)
90 #print(number_of_rows)
91
92 #Printing the beginning part to see if the data is read correctly
93 #df.head()
94

```

```

95 #Printing the beginning part to see if the data is read correctly
96 #df.tail()
97
98 #Testing the labels
99 reviews = df['Text'].values.tolist()
100 labels = df['Sentiment'].tolist() #convert to category
101 #print(reviews[:2])
102 #print(labels[:2])
103
104 #training_sentences, validation_sentences, training_labels, validation_labels
105 training_sentences, validation_sentences, training_labels, validation_labels
106 #this is on creating stratified sample
107
108 #Preprocessing the data using DistilBert for punctuation splitting and word
109 tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')
110
111 tokenizer([training_sentences[0]], truncation=True,
112           padding=True, max_length=128)
113
114 train_encodings = tokenizer(training_sentences,
115                             truncation=True,
116                             padding=True)
117 val_encodings = tokenizer(validation_sentences,
118                           truncation=True,
119                           padding=True)
120
121 train_dataset = tf.data.Dataset.from_tensor_slices((
122     dict(train_encodings),
123     training_labels
124 ))
125
126 val_dataset = tf.data.Dataset.from_tensor_slices((
127     dict(val_encodings),
128     validation_labels
129 ))
130
131 print(val_dataset)
132
133 #tbd
134 model = TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
135
136 optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5, epsilon=1e-08)
137 callbacks=tf.keras.callbacks.EarlyStopping(
138     monitor='accuracy',
139     min_delta=0.0001,
140     patience=3,
141     mode='auto',
142     verbose=2,

```

```

143     baseline=None
144 )
145 model.compile(optimizer=optimizer, loss=model.compute_loss, metrics=['accu
146 model.fit(train_dataset.shuffle(100).batch(16),
147           epochs=2,
148           batch_size=16,
149           validation_data=val_dataset.shuffle(100).batch(16), callbacks=call
150
151 '''
152 import matplotlib.pyplot as plt
153
154 plt.title('Loss curves')
155 plt.plot(model.train_loss_history, '-', label='train')
156 plt.plot(model.val_loss_history, '-', label='val')
157 plt.legend(loc='lower right')
158 plt.xlabel('Iteration')
159 plt.show()
160
161 '''
162
163 model.save_pretrained("./sentiment")
164
165 loaded_model = TFDistilBertForSequenceClassification.from_pretrained("./sen
166
167 import pandas as pd
168 #Testing a model with a user-written input
169
170 #df = pd.DataFrame({'Text': ["This is a not a good product. I hate it", "Th
171 #test_sentence = "This is a not a good product. I hate it"
172
173 df1 = pd.read_csv("/content/drive/MyDrive/summary_test.csv", index_col=0, e
174 df1=df1.loc[0:50]
175 selected_columns = df1[["Summary","Text","Generated_summary"]]
176 df = selected_columns.copy()
177 df = df.dropna()
178
179
180 df.head()
181
182 for i in range(0, len(df)):
183     df['Sentiment_text'] = i
184     df['Sentiment_summary'] = i
185
186 for i in range(0, len(df)):
187
188     predict_input_text = tokenizer.encode(df['Text'][i],
189                                           truncation=True,
190                                           padding=True,

```

```

191         return_tensors="tf")
192     tf_output_text = loaded_model.predict(predict_input_text)[0]
193     tf_prediction_text = tf.nn.softmax(tf_output_text, axis=1)
194     labels = ['Negative', 'Positive']
195     label_text = tf.argmax(tf_prediction_text, axis=1)
196     label_text = label_text.numpy()
197     df["Sentiment_text"][i] = (labels[label_text[0]])
198
199
200 #df = df.append(data, columns = "Sentiment")
201 #print(df['Text'], df['Sentiment_text'])
202
203 for i in range(0, len(df)):
204
205     predict_input_sum = tokenizer.encode(df['Generated_summary'][i],
206                                         truncation=True,
207                                         padding=True,
208                                         return_tensors="tf")
209     tf_output_sum = loaded_model.predict(predict_input_sum)[0]
210     tf_prediction_sum = tf.nn.softmax(tf_output_sum, axis=1)
211     #labels = ['Negative', 'Positive']
212     label_sum = tf.argmax(tf_prediction_sum, axis=1)
213     label_sum = label_sum.numpy()
214     df["Sentiment_summary"][i] = (labels[label_sum[0]])
215
216
217 #print(df['Generated_summary'], df['Sentiment_summary'])
218 df.head()
219
220 tag = 0
221 for i in range(0, len(df)):
222     if (df['Sentiment_text'][i] != df['Sentiment_summary'][i]):
223         tag = tag + 1
224
225 Error = tag/len(df)
226 print(Error)

```

A.3 Set up

A.4 Scores outputs