Data Quality Metrics

(IN COLLABORATION WITH BMW GROUP)

Developers: Vladana Djakovic, Valari Pai, Ekaterina Shmaneva

Supervisors: Dr Maka Karalashvili (ext.), Prof. Dr Matthias Schubert (int.)

CONTENT

MOTIVATION

When (or after) the car is produced, different defects occur. These defects are recorded and stored in the data source – the “Knowledge base” – that summarizes similar defects and assigns them to the prebuilt defect cluster. Each defect contains high amount of human written-text data, which makes analysis time-consuming and complicated

OUR GOAL

To build a model that will process the human created text data of different length, create a summary of it, classify it based on the „sense“ of the generated summary and evaluate the quality of it

Summarization

– a text that is produced from one or more texts, that contains a significant portion of the information in the original text(s).

If it was created with the computer, it is called **automatic summarization.**

Can be **abstractive** and **extractive.**

Classification

– categorizing open-ended text into two or more predefined classes based on some rules or similarities between these texts.

Can be performed based on of the three approaches:

- **Rule-based systems**
- **ML-based systems**
- **Hybrid systems**

Models, used only for summarization

(e.g. Sumy)

Models, used only for classification

(e.g. Naive Bayes, SVMs)

Models, used for both tasks

(e.g. Gensim, CNNs, RNNs, BERT-based models, GPT models, XLNet, T5)

Data access and security issues

Insufficient resources issues

Data access and security issues

(new open-source dataset should be found, that would match the original one)

Data

Amazon Product Review Dataset

Information

10 columns:

Structure

ID, Product ID, User ID, Profile Name, Helpfulness Numerator, Helpfulness Denominator, Score, Time, Summary, Text

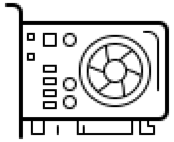
568.427 reviews

Content

2 columns kept:

Useful data

Summary, Text



Lack of proper computational resources

(lightweight models should be found to complete the task)

OUR CHOICE: T5 model

summarization

Encoder & Decoder blocks

(decoder block helps model to create better summary)

The output is a text string

(many other models have labels/spans as output
→ improper output for summarization task)

Robust and extensible

(weights are assigned more properly,
the model can be easily modified to other tasks)

OUR CHOICE: DistilBERT model
classification

Small, fast, cheap

(40% less parameter than BERT → 60% faster)

Distilled & transfer-learning adapted

(mix of the distillation and transfer-learning
→ Above 90% accuracy on classification)

Open-source & flexible

(model available via HuggingFace,
retains 97% of BERT performance)

PROJECT TIMELINE

Oct, 2021

(Getting to know the supervisor, the project and the goal of it, searching for the data)

Nov, 2021

(Exploration of the dataset, metric extraction & processing ideas, building a data loader)

Dec, 2021

(Research on summarization techniques, exploring necessary packages)

Jan, 2022

(First-choice model research, baseline model building (RoBERTa), research on classification)

PROJECT TIMELINE

Feb, 2022

(RoBERTa issue handling, parameter fine-tuning, classification implementation)

Mar, 2022

(Classification model issue handling, testing and parameter fine-tuning)

Apr, 2022

(Second-choice model research and implementation (Google T5 model))

May, 2022

(New model issue handling, parameter fine-tuning, documentation preparation)

Note on summarization model change

	RoBERTa	vs.	Google T5
Pre-training	5 datasets containing about 160GB of text		Multi-task un-&supervised tasks on 16 datasets
Pre-training time	1 day (1024xV100 GPUs, batch size 8k)		~12 hours (Titan RTX, batch size 8k)
Base of the model	BERT (bi-directional transformer model)		–
Parameter set	354M parameters (RoBERTa-large)		11B parameters (t5-11b)
Optimizer used	Adam (learning rate = 0.0006)		AdamW & AdaFactor (learning rate = 0.0003)
ROUGE Score (official paper)	F-measure (ROUGE-L) = 25.67		F-measure (ROUGE-L) = 38.35

Implementation: Set Up

Computational System

Google Colab Pro
(up to 24GB RAM, K80, P100, T4 GPUs)

Environment

Python ver. 3.8.5 and above

Model & documentation

Implementation: T5 Tokenizer

```
class SummaryModel(pl.LightningModule):

 def __init__(self):
   super().__init__()
   #initializing model
   self.model = T5ForConditionalGeneration.from_pretrained(modelName, return_dict=True)


 # Defining forward function and it output
 def forward(self,input_ids, attention_mask, decoder_attention_mask, labels=None):
   output = self.model(
      input_ids,
      attention_mask=attention_mask,
      labels=labels,
      decoder_attention_mask=decoder_attention_mask
    )
   return output.loss, output.logits
```

Implementation: T5 Tokenizer

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

  batch_dictionary={ "loss": loss, "labels": labels}

  self.log("Loss/Train (Batch)", loss, prog_bar=True,logger=True)
  self.logger.experiment.add_scalar("Loss/Train (Epoch)", loss, self.current_epoch)
  #return loss
  return batch_dictionary
```

Implementation: T5 Tokenizer

```
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.logger.experiment.add_scalar("Loss/Val (epoch)",loss,self.current_epoch)
  self.log("Loss/Val (Batch)", loss, prog_bar=True,logger=True)
  epoch_dictionary={'loss': loss}
  return epoch_dictionary
```

Implementation: T5 Tokenizer

```
  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.logger.experiment.add_scalar("Loss/Test",loss,self.current_epoch)
    self.log("test_loss", loss, prog_bar=True,logger=True)
    return {'loss': loss}

 # Configurating optimizer as most used one AdamW
  def configure_optimizers(self):
     return AdamW(self.parameters(), lr=0.0001)
```

Implementation: DistilBERT Tokenizer

```
tokenizer([training_sentences[0]], truncation=True, padding=True, max_length=128)
    # Tokenizing the data
train_encodings = tokenizer(training_sentences,truncation=True,padding=True)
val_encodings = tokenizer(validation_sentences,truncation=True,padding=True)
# Slicing the dataset
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))

# Loading the DistilBert model from transformers
model = TFDistilBertForSequenceClassification.from_pretrained
   ('distilbert-base-uncased',num_labels=2)

# Defining and fitting the model on the training data
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=['accuracy'])
model.fit(train_dataset.shuffle(100).batch(16),
          epochs=5,
          batch_size=16,
          validation_data=val_dataset.shuffle(100).batch(16),callbacks=callbacks)
```

Implementation: DistilBERT Tokenizer

```
tokenizer([training_sentences[0]], truncation=True, padding=True, max_length=128)
    # Tokenizing the data
train_encodings = tokenizer(training_sentences,truncation=True,padding=True)
val_encodings = tokenizer(validation_sentences,truncation=True,padding=True)
# Slicing the dataset
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))

# Loading the DistilBert model from transformers
model = TFDistilBertForSequenceClassification.from_pretrained
   ('distilbert-base-uncased',num_labels=2)

# Defining and fitting the model on the training data
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=['accuracy'])
model.fit(train_dataset.shuffle(100).batch(16),
          epochs=5,
          batch_size=16,
          validation_data=val_dataset.shuffle(100).batch(16),callbacks=callbacks)
```

Implementation: DistilBERT Classes

```
# 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]])
```

Initial text of the review

Implementation: DistilBERT Classes

```
# 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]])
```

Newly generated summary

Performance: Metrics

Running time

shows the amount of time that was required to perform the training (only)

Validation and training losses

describe the performance of the model, indicating how well it is fitting the training and the new data correspondingly

ROUGE Score

compares automatically produced summary against reference (human-written) ones

Accuracy

defines the number of correctly predicted data points out of all the data points

Performance: T5

	10k sample	vs.	100k sample
Running time	approx 30 min		approx 3 hours
Train loss	2.481		2.849
Validation loss	3.532		3.859
ROUGE Scores			
Rouge-1			
recall	0.266		0.143
precision	1.0		1.0
f-measure	0.421		0.250
Rouge-L			
recall	0.266		0.143
precision	1.0		1.0
f-measure	0.421		0.250

Performance: Classification

	10k sample	vs.	100k sample
Running time	approx 36 min		approx 3.5 hours
Train loss	0.137		0.094
Accuracy	0.953		0.963
Validation loss	0.274		0.194
Validation accuracy	0.901		0.935
Classification error	0.1		0.1

FUTURE

1. Model adaptation to the BMW data
2. Further summarization model fine-tuning to make the model more precise
3. Expanding the classi-fication of the data (based on the information, that the summaries contain)

CONCLUSIONS

I

Why task is important

(Analysing the human-written defects is not easy and time consuming)

II

What models exist

(*Summarization*: BERT (&variations), GPT, T5, CNN, RNN; *Classification*: Naive Bayes, SVM, summarization ones)

III

What models were chosen

(*Summarization*: Google T5 model; *Classification*: DistilBERT)

IV

Performance analysis

(*Summarization*: all of the n-grams in the generated summaries are present in the reference text; *Classification*: overall accuracy > 90%, error = 0.1)

V

What else can be done

(*Summarization*: fine-tuning to increase the quality; *Classification*: expanding the number of classes)

CONCLUSIONS

I **Why task is important**
(Analysing the human-written defects is not easy and time consuming)

II **What models exist**

III **What model we‘ve chosen**

IV **Performance analysis**

V **What else can be done**


THANK YOU

References

Literature:

- Jacob Devlin andothers. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
  http://arxiv.org/abs/1810.04805
- Aastha Singh. Evolving with BERT: Introduction to RoBERTa
  https://medium.com/analytics-vidhya/evolving-with-bert-introduction-to-roberta-5174ec0e7c82
- Yinhan Liu andothers. RoBERTa: A Robustly Optimized BERT Pretraining Approach
  https://arxiv.org/abs/1907.11692
- Rohan Jagtap. RoBERTa: Robustly Optimized BERT-Pretraining Approach. DataSeries
  https://medium.com/dataseries/roberta-robustly-optimized-bert-pretraining-approach-d033464bd946
- Anubhav. Step by Step Guide: Abstractive Text Summarization Using RoBERTa
  https://anubhav20057.medium.com/step-by-step-guide-abstractive-text-summarization-using-roberta-e93978234a90
- Manmohan Singh. Summarize Reddit Comments using T5, BART, GPT-2, XLNet Models
  https://towardsdatascience.com/summarize-reddit-comments-using-t5-bart-gpt-2-xlnet-models-a3e78a5ab944
- Sukanya Bag. Text Summarization using BERT, GPT2, XLNet
  https://medium.com/analytics-vidhya/text-summarization-using-bert-gpt2-xlnet-5ee80608e961
- Summarization with GPT-3. KDnuggets. Section: Products and Services
  https://www.kdnuggets.com/2022/04/packt-summarization-gpt3.html
- Priya Shree. The Journey of Open AI GPT models. Walmart Global Tech Blog
  https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2
- Maria Yao. 10 Leading Language Models For NLP In 2021
  https://www.topbots.com/leading-nlp-language-models-2020/
- Qiurui Chen. T5: a detailed explanation
  https://medium.com/analytics-vidhya/t5-a-detailed-explanation-a0ac9bc53e51
- Pedro Marques. Fine tuning a T5 text-classification model on colab. Pedrormarques
  https://pedrormarques.wordpress.com/2021/10/21/fine-tuning-a-t5-text-classification-model-on-colab/

References

Literature:

- Mathew Alexander. Data to Text generation with T5; Building a simple yet advanced NLG model
  https://towardsdatascience.com/data-to-text-generation-with-t5-building-a-simple-yet-advanced-nlg-model-b5cce5a6df45
- Abstractive Summarization Using Google's T5. Turbolab Technologies Blog. Section: Technology
  https://turbolab.in/abstractive-summarization-using-googles-t5/
- Eduard Hovy. Text Summarization. The Oxford Handbook of Computational Linguistics
  https://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199276349.001.0001/oxfordhb-9780199276349-e-32
- Praveen Dubey. Understand Text Summarization and create your own summarizer in python
  https://towardsdatascience.com/understand-text-summarization-and-create-your-own-summarizer-in-python-b26a9f09fc70
- Text Classification: What it is And Why it Matters
  https://monkeylearn.com/text-classification/
- Shrivar Sheni. Text Summarization Approaches for NLP – Practical Guide with Generative Examples
  https://www.machinelearningplus.com/nlp/text-summarization-approaches-nlp-example/
- Susan Li. Multi-Class Text Classification with Doc2Vec & Logistic Regression
  https://towardsdatascience.com/multi-class-text-classification-with-doc2vec-logistic-regression-9da9947b43f4
- In´es Rold´os. Go-to Guide for Text Classification with Machine Learning. MonkeyLearn Blog. Section: Text Classification
  https://monkeylearn.com/blog/text-classification-machine-learning/
- Leo Laugier. Extractive Document Summarization Using Convolutional Neural Networks Reimplementation
  https://www.semanticscholar.org/paper/Extractive-Document-Summarization-Using-Neural-Laugier/ed0f189bbbbccceefb41ccb36e1c5b62bc36d2fb
- Christian Heumann andothers. LMU Course: Deep Learning for Natural Language Processing
  https://moodle.lmu.de/course/view.php?id=17645
- Zhenzhong Lan andothers. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations
  https://arxiv.org/abs/1909.11942
- Victor Sanh andothers. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter
  https://arxiv.org/abs/1910.01108

References

Literature:

- Zihang Dai. XLNet: Generalized Autoregressive Pretraining for Language Understanding
  https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1606199&dswid=-4773
- What is a Data Steward? — Experian Business
  https://www.experian.co.uk/business/glossary/data-steward/
- SAS-institute. What is a data scientist?
  https://www.sas.com/en_us/insights/analytics/what-is-a-data-scientist.html
- Rachel Draelos. Best Use of Train/Val/Test Splits, with Tips for Medical Data
  https://glassboxmedicine.com/2019/09/15/best-use-of-train-val-test-splits-with-tips-for-medical-data/
- Andrew Fogarty. Summarization: T5
  http://seekinginference.com/applied_nlp/T5.html#rouge-metrics
- Kavita Ganesan. An intro to ROUGE, and how to use it to evaluate summaries
  https://www.freecodecamp.org/news/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840/

Data:

- Amazon. Amazon Product data
  https://jmcauley.ucsd.edu/data/amazon/

Imagery:

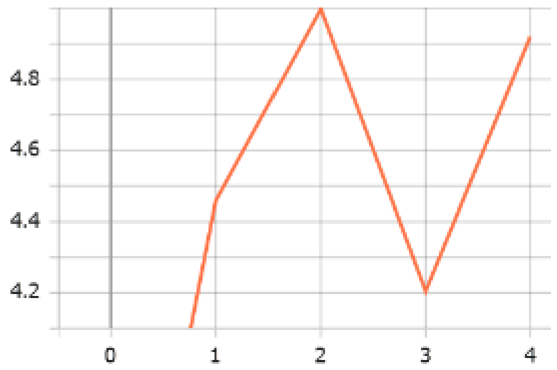
- unsplash.com
- pinterest.de
- behance.net

Graphics:

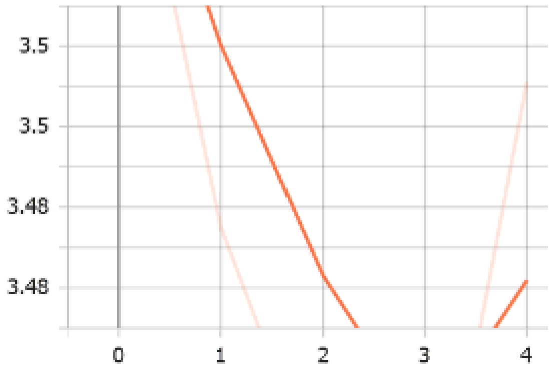
- icons8.com

Performance analysis: T5 Graphs

10k sample

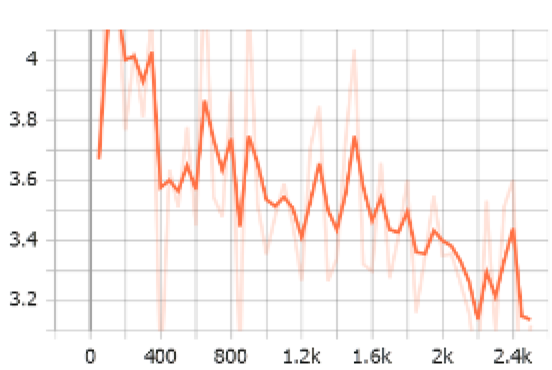


(a) *Train loss calculated epoch-wise*

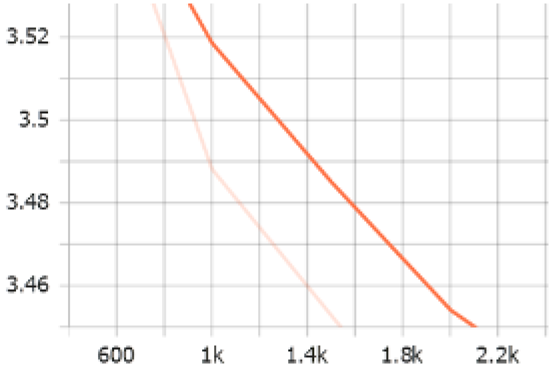


(b) *Validation loss calculated epoch-wise*

100k sample



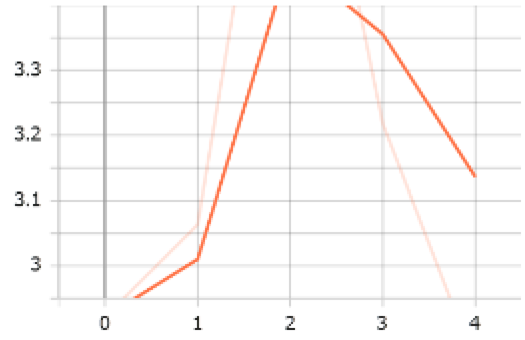
(a) *Train loss calculated batch-wise*



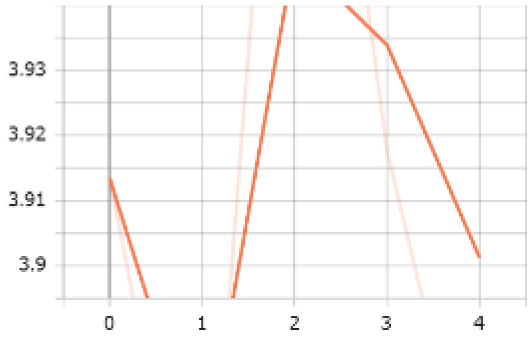
(b) *Validation loss calculated batch-wise*

Performance analysis: T5 Graphs

10k sample

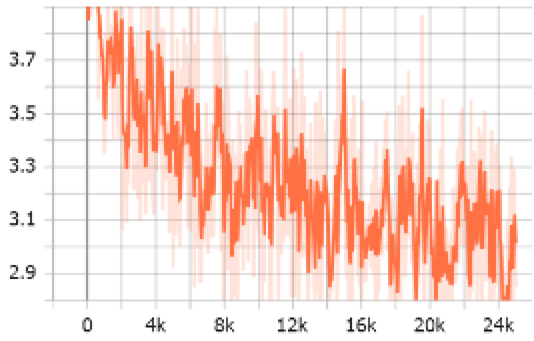


(a) *Train loss calculated epoch-wise*

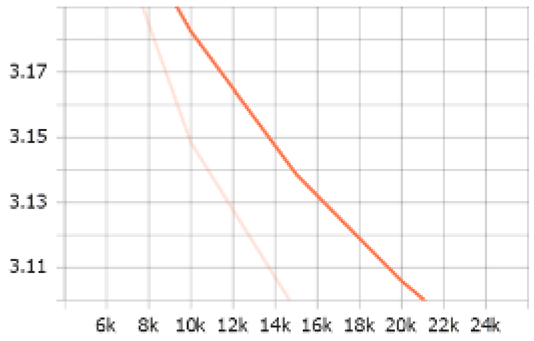


(b) *Validation loss calculated epoch-wise*

100k sample



(a) *Train loss calculated batch-wise*



(b) *Validation loss calculated batch-wise*