Anita Kurm: 201608652
Maris Sala: 201604882

Data Science for MSc Cognitive Science
Aarhus University, May 2020

LARGE BERT VS. DISTILBERT: FIND THE RIGHT ANSWER

HTTPS://GITHUB.COM/MARISSALA/DATA-SCIENCE-BERT

ANITA KURM & MARIS SALA

Students' individual contributions are marked with their initials in square brackets: [A] and [M]. Abstract, introduction and conclusion were written together.

ABSTRACT

In this paper the large version of BERT (Bidirectional Encoder Representations from Transformers by Google AI) is compared to its smaller knowledge-distilled version DistilBERT. Both models were fine-tuned to the SQuAD 1.1 task. The aim was to see how well these pre-made models perform on a new dataset that hasn't been part of their training nor fine-tuning data. For applications in the real world, it's ideal to find a model that doesn't take extra training and fine-tuning on large datasets to give accurate results with a small computational cost. To evaluate this, we compared the running time and performance of both models in a Question Answering task on the TweetQA dataset. GLEU and METEOR scores were used and optimized for automated performance evaluation. We found that DistilBERT ran 529% faster and reached around 87-90% of large BERTs performance, even though it was based on a smaller base-BERT. The possible applications and future model improvements are considered in the discussion.

Contents

INTRODUCTION [A+M]

In 2018, Devlin et al. from the Google AI Language team released what is currently considered the state-of-the-art language model developed so far - BERT (Bidirectional Encoder Representations from Transformers). The model represents a stack of encoders from the Transformer neural net architecture proposed by Vaswani et al. (2017) with an additional output layer, that can be tailored to the language task at hand. Since its release date, Google AI's pre-trained version of BERT has been successfully fine-tuned to a variety of datasets and language tasks, surpassing other language models in performance.

BERT reaches high levels of accuracy, but it also takes a lot of computing power to fine-tune and run, since it has around 340M parameters. In evaluating a language model's performance, optimizing for accuracy of predictions is necessary. However, it also matters that the model can run on various devices without being restricted to the most powerful hardware, making model compression research an important domain in Data Science. As researchers explored different ways to compress BERT without undermining its performance, they published different compressed versions of BERT in a variety of sizes and with different tasks in mind.

Our focus is to compare a large BERT model to a knowledge-distilled version of BERT in a Question Answering task – a challenging language task less prominent in the literature on BERT. Both models would be tested on a new dataset that the models haven't been trained on previously. We are both interested in inference time and how well the results compare to each other and to the golden standard based on the appropriate evaluation metrics. Having an overview of how well the models handle a known task with an unknown dataset would give us an idea of how generalizable the models are and how much both would need to be improved.

Our prior expectation is that a compressed model would yield shorter inference time but worse performance than a larger model. Our main research question is to evaluate how close the performance of a compressed model can come to its larger counterpart, and whether that difference is justified by lower computational costs.

THEORY

OVERVIEW OF BERT [A]

The original BERT by Devlin et al. (2018) was composed of 24 Encoder blocks from the Transformer architecture introduced by Vaswani et al. (2017). The Transformer architecture is described best in the original paper (Vaswani et al., 2017) and will not be included in this overview.

The standard input to BERT is a sequence of tokens, including special tokens marking start of the sequence and breaks between the sentences. All tokens of the sequence then undergo an embedding process shown in Figure 1, which is similar to input embedding stage in the original Transformers but also captures an additional segment embedding to track sentence affiliation.

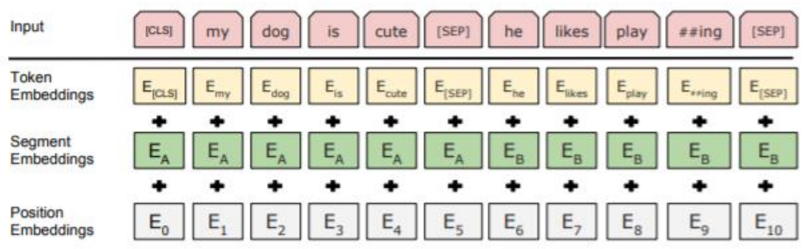

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Figure 1. Input to BERT and summarization of its embeddings (image from Devlin et al. (2018)). Token (WordPiece embeddings), segment and position embeddings are summed together

Resulting input embeddings then flow through BERT's architecture, which according to the structural and qualitative analysis by Tenney et al. (2019) mimics a traditional NLP pipeline. There, earlier layers of the network seemed to conduct lower-level operations (e.g. part of speech tagging and parsing-like processes), while higher-level processes (e.g. assignment of semantic roles and coreference) seemed to be localised in later layers of BERT and were used by the model to revise lower-level decisions in case of ambiguity.

On top of the encoder stack, BERT's architecture ends with the final output layer. What this layer should look like depends on the particular language task at hand and is therefore defined for every task separately in the pre-training or fine-tuning process.

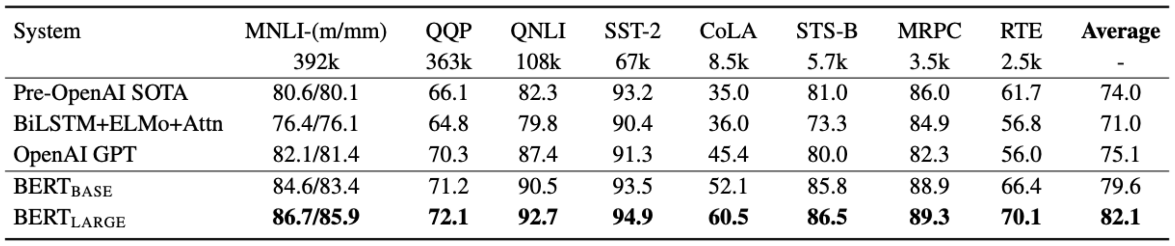
Unlike many unidirectional language models, BERT is pre-trained to produce a word's representation that captures its context in both directions. This is achieved by setting a 'masked language model' objective as the first part of the model's pre-training procedure. By randomly masking 15% of tokens in the input sequence and training the model to guess those missing tokens, parameters on all levels of the model adjust to capture the token's context regardless whether it's on the right or on the left of the missing token (Devlin et al., 2018).

The second part of BERT's pre-training is the 'next sentence prediction task' performed simultaneously with the 'masking task'. This further adjusts model weights to also capture the relationship between two consequent sentences in the text. Just as masking, this task can be performed unsupervised, which allows pre-training on large and rich language corpora without imposing too many additional costs to the researcher. The original BERT was pre-trained on texts from BooksCorpus (800M words) and English Wikipedia (2,500M words).

The resulting pre-trained BERT can be used as a strong starting point for different language tasks regardless of their scale and domain (e.g. single word prediction, question answering, general language understanding) (Devlin et al., 2018).

While being faster than a convolutional or recurrent neural network of a similar size, the original large BERT by Devlin et al. (2018) is still considered to be a computationally expensive model with 340M parameters. The smaller version presented in the same paper was base BERT with 110M parameters. Both models scored well in a wide array of language tasks (see Table 1).

Table 1. Performance of large and base BERT on GLUE benchmark was better than in other models. Table from Devlin et al. (2018).



System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT$_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

MODEL COMPRESSION [A]

There are several ways trained neural net models can be compressed to gain an advantage in computational costs. When models are trained, they are usually over-parameterized because finding the appropriate parameters can be very difficult and it's hard to train an appropriately parameterized model using gradient descent (Gordon, 2020). Instead, a large model is compressed to a simpler one by removing its redundancies and thus bringing it closer to an appropriately parameterized model. In addition to a lessened computational cost, compressed models tend to be more generalizable to noisy data and thus it makes sense to expect them to perform better on new data (Vijay, 2019).

Different types of model compression techniques focus on a different kind of property that differentiates a large model from a compressed one: redundancies such as many of the weights approximating 0 or some of the layers learning to do similar functions (Table 2).

Table 2. Types of model compression. Table from Gordon (2020).

Compression Method	**Redundancy**	**Why?**	**During Training**
Weight Pruning	Many zero weights	Unclear	Rigged Lottery
Weight Sharing	Different layers tend to perform similar functions	Images are locally coherent/compositional. So are sentences.	Convolutions, Recurrences, ALBERT
Quantization	Weights are low-precision	Unclear	XNOR Net, DoReFa-Net, ABC-Net
Matrix Factorization	Matrices are low rank	Dropout, Nuclear-norm regularization	ALBERT
Knowledge Distillation	Unclear	Unclear	BANs, Snapshot KD, Online KD

Table 2 illustrates various methods of model compression, which kinds of model redundancy they take advantage of and why. For knowledge distillation it is unclear which redundancy is used. We choose to focus on this methodology since it is interesting to study and experiment with a technique that performs well but the underlying reasons remain unclear.

KNOWLEDGE DISTILLATION [M]

Model compression in the form of knowledge distillation was originally proposed by Buciluǎ et al., (2006). However, mostly a more recent generalization of the topic published by Hinton et al. (2015) was used by Sanh et al. (2019) to distil base BERT into DistilBERT.

Knowledge distillation is a method where a smaller network (student) is taught by a larger network (teacher) about how to do what the trained network can do (Figure 2). In a supervised learning task, a deep neural net classifier model would be able to tell the difference between the categories and even if it makes mistakes, it makes likelier mistakes over less likely ones. Conceptually speaking, this kind of knowledge has been referred to as dark knowledge (Hinton et al., 2015) and having a good grasp of it means that the model is good at generalizing, not just plainly classifying.

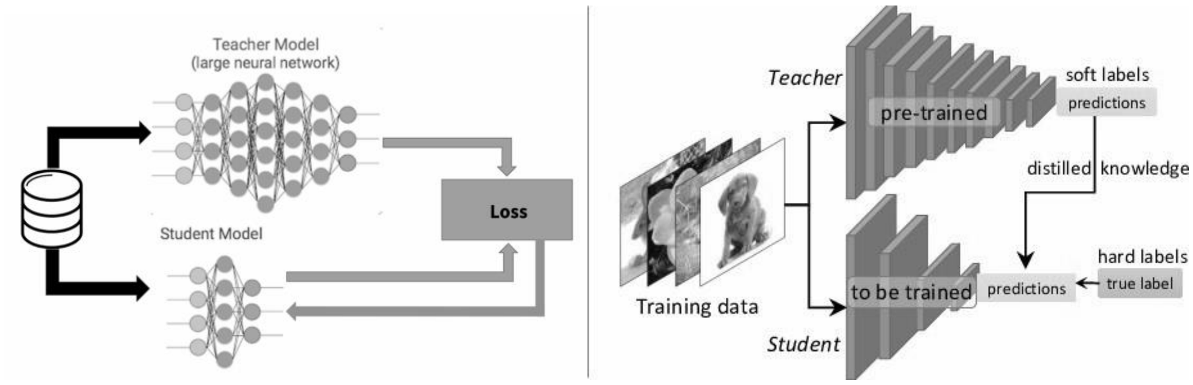


Figure 2. The interactions between a teacher model teaching the student model in a step-by-step fashion (Ganesh, 2019).

Machine learning algorithms use different loss functions to find out which answer is closest to the true label. In training of classification models, the categories are referred to as “hard targets” where the correct category is noted with 1 and the others with 0s such as [0, 1, 0, 0]. “Soft targets” represent the same categories as distribution of probabilities, such as [0.10, 0.60, 0.15, 0.15]. Usually deep neural networks are trained with a cross-entropy over the hard targets, but in the transfer of knowledge the student is instead trained with a cross-entropy of the soft targets retrieved from the teacher model. The training loss is then defined as:

$$L = -\sum_i t_i * log(s_i)$$

Where *t* is the logits from the teacher and *s* the logits of the student (Sanh et al., 2019). This is how deep knowledge can be practically transferred between the two models. Large models such as the teacher are good at generalizing due to their size and parameter amount. Transferring dark knowledge helps the student model be just as good at generalization compared to a model of the same size (Zhang, 2014).

The soft targets probability distributions are calculated with the following softmax activation function:

$$q_i = \frac{exp(z_i/T)}{\Sigma_j exp(z_j/T)}$$

Where q_i is the probability of a class, $z_i$are the logits for each class for every datapoint which are converted to probabilities using T as the temperature. At the time of training, a higher T

value, which results in softer distributions, is used in both the teacher and the student model because it allows for better deep knowledge understanding of the data. After training and during inference, *T* is set back to 1 (Sanh et al., 2019; Vijay, 2019).

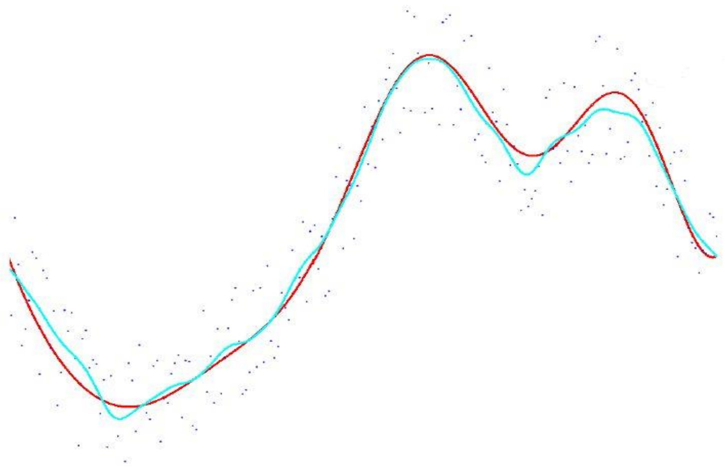


Figure 3. The teacher model (blue line) compared to the student model (red line) in generalizing on the complex dataset. Figure from Vijay (2019).

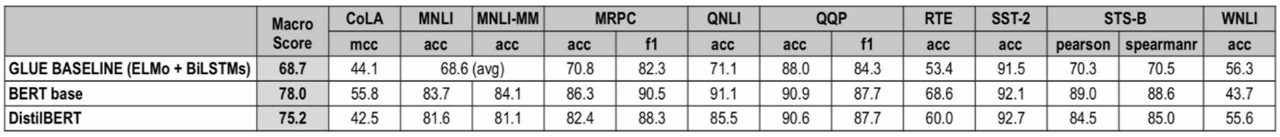
Figure 3 shows the teacher model (blue line) in comparison to the smoother student model (red line). The student model takes the geometric mean of all individual predictive distributions for the soft targets. High entropy (surprisal) in soft targets provides the training process with more information and less variance. This improves student model's ability to replicate the behavior of its teacher model in comparison to a small model that was directly trained on the same or smaller dataset. The compressed model can also use a higher learning rate and thus be fine-tuned faster than its teacher (Vijay, 2019).

KNOWLEDGE DISTILLED MODELS BY HUGGINGFACE [M]

HuggingFace has made several distilled models of BERT, the first one being DistilBERT (HuggingFace, 2020a; Sanh et al., 2019). DistilBERT has the same architecture as BERT but it has less layers, lacks the token-type embeddings and the pooler which is used for the next sentence prediction task. Because both BERT and DistilBERT have the same number of hidden layers, it was possible to initialize it's training by using a hidden layer from its teacher. This helps the model converge (as opposed to "the lottery ticket" hypothesis as described in Frankle & Carbin, 2019). DistilBERT was trained similarly to BERT on the same dataset with similarly large batch sizes and masking.

HuggingFace reports DistilBERT to have half the parameters of BERT base and yet based on the GLUE benchmark, it performs with 95% accuracy of the BERT base as seen in Table 3. Both base BERT and DistilBERT performed better than the GLUE baseline.

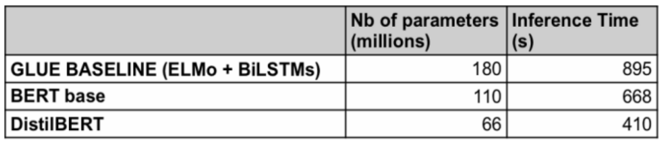
Table 3. Comparison of language models on different GLUE benchmark tests. Table from Sanh et al. (2019)



	Macro Score	CoLA	MNLI	MNLI-MM	MRPC		QNLI	QQP		RTE	SST-2	STS-B		WNLI
		mcc	acc	acc	acc	f1	acc	acc	f1	acc	acc	pearson	spearmanr	acc
GLUE BASELINE (ELMo + BiLSTMs)	**68.7**	44.1	68.6 (avg)		70.8	82.3	71.1	88.0	84.3	53.4	91.5	70.3	70.5	56.3
BERT base	**78.0**	55.8	83.7	84.1	86.3	90.5	91.1	90.9	87.7	68.6	92.1	89.0	88.6	43.7
DistilBERT	**75.2**	42.5	81.6	81.1	82.4	88.3	85.5	90.6	87.7	60.0	92.7	84.5	85.0	55.6

Table 4 shows that due to its much smaller size, DistilBERT is more than 60% faster than BERT, and 120% faster than ELMo + BiLSTM.

Table 4. Comparison of the language models in terms of parameters and inference time. Table from Sanh et al. (2019)

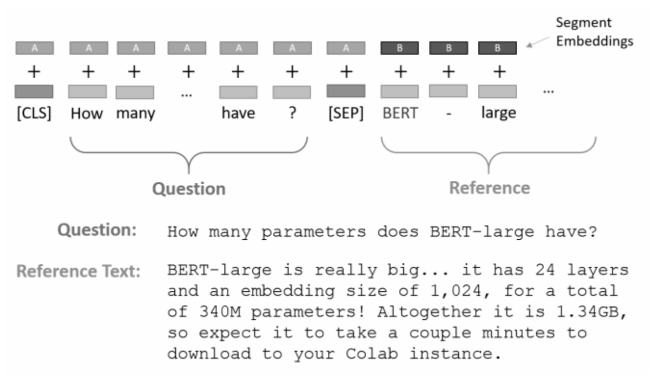


	Nb of parameters (millions)	Inference Time (s)
GLUE BASELINE (ELMo + BiLSTMs)	180	895
BERT base	110	668
DistilBERT	66	410

HuggingFace has made other distilled versions of BERT such as DistilmBERT for multilingual tasks and DistilRoBERTa which has been distilled from RoBERTa - another version of BERT. Different distilled models aim to address issues relevant for specific tasks, e.g. handling non-English datasets. These models are however not available in their fine-tuned form in HuggingFace's Transformers package.

BERT FOR QUESTION ANSWERING [A]

BERT can handle a wide variety of language tasks, but the one in the focus of this paper is the Question Answering (QA) task. In most cases the task is based on applying BERT to the Stanford Question Answering Dataset (SQuAD), where given a question and a prompt which includes the answer, the model is able to highlight the area in the prompt that corresponds to the answer to the question (McCormic, 2020; Figure 4).



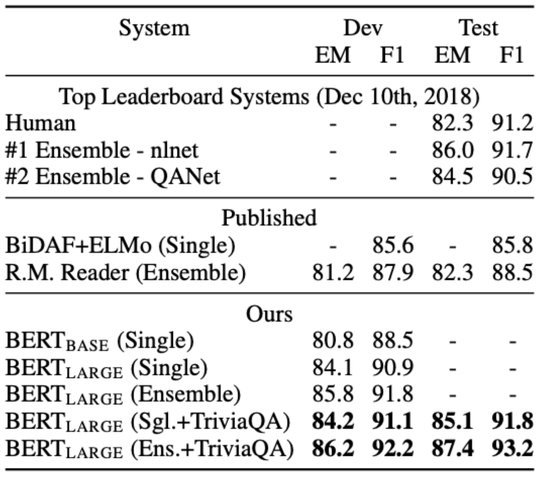
Question: How many parameters does BERT-large have?

Reference Text: BERT-large is really big... it has 24 layers and an embedding size of 1,024, for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple minutes to download to your Colab instance.

Figure 4. The premise on which BERT highlights the correct answer to a question in the reference text. Figure from McCormic (2020).

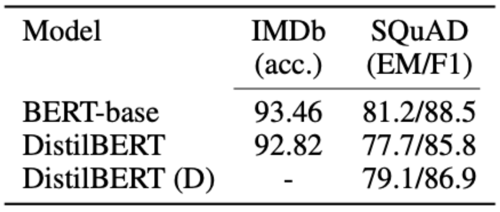
SQuAD 1.1. task's results presented Table 5 by Devlin et al. (2018) suggest that large and base BERT can be very successful in the QA task, given appropriate fine-tuning. As can be seen in Table 6 by Sanh et al. (2019), DistilBERT can perform almost as successfully as base BERT. Given that base BERT's performance is identical in both tables, we can estimate that DistilBERT was 96.9% as good as base BERT and 94.8% as good as large BERT.

Table 5. Comparison of language models to human performance on SQuAD 1.1. With additional fine-tuning on TriviaQA dataset, large BERT outperformed even human judgement. Table from Devlin et al. (2018).



System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
BERT$_{BASE}$ (Single)	80.8	88.5	-	-
BERT$_{LARGE}$ (Single)	84.1	90.9	-	-
BERT$_{LARGE}$ (Ensemble)	85.8	91.8	-	-
BERT$_{LARGE}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
BERT$_{LARGE}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 6. Comparison of DistilBERT to base BERT by performance on SQuAD 1.1. DistilBERT achieves close accuracy. Table from Sanh et al. (2019).



Model	IMDb (acc.)	SQuAD (EM/F1)
BERT-base	93.46	81.2/88.5
DistilBERT	92.82	77.7/85.8
DistilBERT (D)	-	79.1/86.9

This task was chosen by us because most BERT-related literature focuses on other tasks, since datasets for studying automated QA are not as abundant as classification and other kinds of NLP datasets. The QA task is especially interesting to use BERT on because BERTs architecture and training is optimizing towards bidirectional context cues which matters a lot for question answering and dialogue tasks.

AUTOMATED EVALUATION METRICS [M]

Language models are developed to solve language tasks as well as or better than humans, with large gains in speed. To properly measure performance of language models, it is necessary to carefully compare the different metrics that have been made for evaluation purposes. The evaluation of language models should be just as automated and approximate human levels of evaluation quality.

The field of machine translation (MT) has the most automated metrics developed so far. A good evaluation metric needs to fit with the task that it is used for, and when using an evaluation metric, one must be aware of its downsides and take them into consideration when inspecting the results.

In comparing the MT outputs, it's important that the meaning is translated from one language to another. This means that it matters to check for example grammar, synonyms, brevity of the output, word order. One of the first and most generally used metric was developed by IBM and is called the BLEU (bilingual evaluation understudy) (Papineni et al., 2002). BLEU is an intuitive measure of distance between the word matches of the translation to the reference texts. It's designed to work best on corpus-level, and on sentence-level analysis it might give inflated scores (Tatman, 2019). The scores are set from 0 to 1, where 1 signifies a perfect match between the human-made translation and the MT.

In the QA task, the correct answer that the models need to compare to, are given in a human-made text prompt. The models don't generate text themselves, meaning the outputs do not need to be checked for grammar nor synonyms. BLEU is a good fit here because it doesn't penalize for grammatical errors, but it does penalize for an answer being much longer or shorter than the reference. This is due to the way MT sometimes results in more words being produced (Papineni et al., 2002).

A general downside with BLEU is that in translation there are many ways to translate the same sentence, so having a high BLEU score does not entirely correlate with best translation. But in this task, the correct answers are only understandable in one way and the two models work based on highlighting the correct answer in the prompt text. This means that BLEU's main downside does not affect our results.

METEOR (Metric for Evaluation of Translation with Explicit Ordering) is another evaluation tool used to assess quality of MT. It is more complex than BLEU and is considered to approximate human judgement better since it also considers synonyms and uses a stemmer (Lavie & Agarwal, 2007). It weighs recall (matched unigrams divided by unigrams in reference) as more important than precision. METEOR is also adjusted to be used on sentence-level.

Given the nature of the QA task, BLEU and METEOR scores are good to use together since they are set on the same scale and might partially compensate for each other's drawbacks. Their performance should be evaluated within the context of the task to get a better sense of how well they describe the accuracies of the models.

HYPOTHESES AND THE RESEARCH QUESTION [A + M]

Thorough reports by Devlin et al. (2018) and Sanh et al. (2019) give us a good idea of BERT's and DistilBERT's QA performance on a standard dataset like SQuAD 1.1. Given that both models are openly available along with pre-trained and fine-tuned weights for the QA task, it is interesting to see how 'ready-to-use' they are on new data. Language models can make a difference in a variety of fields, where people working with the models would have an easier time if the models would be readily available and able to perform on similar levels as humans.

Question Answering task is a supervised learning problem, and often requires human workers to generate new question answering data (question-answer pairs based on a given prompt). Given how costly such data might be, it is important to see model's baseline performance before investing data into model fine-tuning. We want to understand the BERT's baseline performance with minimal computational and time costs. Baseline performance is considered to be using a pre-trained model on a new dataset which is different both from what BERT was originally trained on and what it was fine-tuned with. Since one of the goals of BERT is to be generalizable across a variety of language tasks, we expect it to perform well even on a dataset it has never seen before.

BERT is computationally expensive to both train and fine-tune, and its inference time might be too long for tasks that need closer to real-time results. BERT is also large which means that it might be difficult to run on less powerful devices. Therefore, gauging whether DistilBERT can be successfully used instead of its larger counterpart is another focal point of this paper.

To summarize, our research question is two-fold:

1) How close can the baseline QA performance of a knowledge distilled model come to baseline performance of its much larger counterpart?
2) Can that difference in performance be justified by reduced computational costs?

Based on the presented theoretical overview, we state the following hypotheses of interest:

1) A knowledge distilled BERT will be at least 60% faster than the large model.
2) A knowledge distilled model will perform worse than the large model, with the difference in evaluation metric being at the very least 5%.
3) Judging from baseline performance on new data, a knowledge distilled BERT can be a better choice than a large BERT model if difference in performance does not exceed 15% while saving at least 60% of inference time.

We gave arbitrarily larger margins for acceptable performance difference to account for potential noisiness of a novel dataset.

METHODS

SELECTED MODELS [A]

As mentioned in the theoretical overview, a large selection of pre-trained versions of BERT can be found in the official documentation for Transformers by HuggingFace (2020)[1]. From that selection, out of models fine-tuned to Question Answering, we chose large BERT pre-

[1] HuggingFace documentation for Transformers (accessed 23.05.2020): https://huggingface.co/transformers/index.html

trained by Devlin et al. (2019) and DistilBERT distilled by Sanh et al. (2020). Our choice was motivated by the fact that DistilBERT is usually compared to its teacher – base BERT, and not the large BERT. We were interested in comparison between one of the lightest well-performing BERTs to one of the most expensive BERTs out there. The difference in model size between chosen models can be seen in Table 7.

Table 7. Model size comparison between large BERT and DistilBERT. (HuggingFace, 2020c)

	Large BERT	**DistilBERT**
Layers	24	6
Dimensions	1024	768
Attention Heads	16	12
Total parameters	340M	66M

Both large BERT by Devlin et al. (2019) and DistilBERT by Sanh et al. (2020) were pre-trained on texts from BooksCorpus (800M words) and English Wikipedia (2,500M words) and then further fine-tuned on the SQuAD task (Rajpurkar et al., 2016) for Question Answering.

To gauge baseline performance, we decided to run pre-trained models as they come on a new dataset without any additional fine-tuning.

DATASET [A]

The dataset we chose for this paper – TweetQA[2] – was developed by (Xiong et al., 2019) for automated social media-based question answering (SoMe QA). As pointed out by dataset's authors, SoMe QA is different from standard QA, as prompts are much shorter and noisier and questions may require understanding of Twitter metadata (e.g. usernames, hashtags, posting date). This motivated the dataset's authors to crowdsource question-answer pairs based on a selection of informational tweets scraped from CNN and NBC websites. The resulting dataset had 13,757 question-answer pairs, out of which 1,979 did not contain publicly available gold

[2] The TweetQA website with short demo of data and the download link (accessed 23.05.2020): https://tweetqa.github.io/

standard answers, as they were reserved as a test dataset for model evaluation in the ongoing competition at CodaLab[3].

In the original paper by Xiong et al. (2019), even after fine-tuning base BERT using the training part of the dataset, TweetQA task was more challenging for the model than standard question answering tasks, like the Wikipedia-based SQuAD task by Rajpurkar et al. (2016). Since our goal was to evaluate baseline performance of pre-trained BERT and DistilBERT in the TweetQA task, we utilised all parts of TweetQA dataset that already contained gold standard answers (merged 'training' and 'development' data, the total of 11,778 question-answer pairs). Short summary of the resulting dataset in comparison to the original dataset can be seen in Appendix 1.

MODEL APPLICATION PROCESS [A]

We decided to conduct data processing on a laptop with 2,3 GHz Intel Core i5 processor and 8 GB RAM, so the tracked processing time would reflect use of models on a personal computer without using GPU or any cloud computing services.

To apply the selected pre-trained models to the TweetQA dataset, we first loaded both models from the Transformers platform. Thorough documentation of the Transformers Python package and openly available OpenAI's pretrained model weights (HuggingFace, 2020b) allows easy access to pre-trained and fine-tuned models (see Code in Appendix 5). Model loading time was tracked for both models.

Since reference texts and questions need to be tokenized before being fed to the question answering model, we also load tokenizers for both models. Notably, even though DistilBERT model class has its own tokenizer called DistilBertTokenizer to be consistent with the rest of the package, it actually still uses standard BERT tokenizer in the backend (HuggingFace, 2020a). This means that both models use identical tokenizers, but they are fetched using different model-specific commands and thus their loading time will be tracked separately.

We then wrote a function that would conduct question answering using a specified model and track model's answer prediction and processing time for every question in the dataset.

[3] Competition at Codalab (accessed 23.05.2020): https://competitions.codalab.org/competitions/20307?secret_key=6684a0cf-9eac-4ac1-a648-213a3961e0fc

AUTOMATED EVALUATION METRICS SELECTION [M]

As BLEU performs best on corpus-level, Google developed their own adjusted form of BLEU which is better at assessing outputs on sentence-level - GLEU score (Wu et al., 2016). GLEU score is computed similarly to BLEU, recall is matching n-grams divided by total n-grams in the golden standard, and precision is the matching n-grams divided by total n-grams in the output. GLEU score is then calculated as the minimum of recall and precision.

To use METEOR score in a way that optimizes more towards the task at hand, the parameters in METEOR were adjusted according to suggestions made by Lavie & Agarwal (2007). They adjusted parameters to optimize for adequacy, fluency, and a sum of both. We chose the parameters that optimize both in order to remain conservative and try to reach a win-win scenario of output evaluation accuracy (see Appendix 2).

The parameter α controls relative weights for precision and recall. The new α defined in our evaluation is lower than the original which means it sets a smaller weight on precision. β controls the shape of penalty as a function of fragmentation, and γ is the relative weight for penalty of fragmentation. Fragmentation helps check whether the order of words is close to the true order. Both β and γ are lower than they were originally, meaning that the penalty is also lower than before (Lavie & Agarwal, 2007). Both GLEU and METEOR scores are calculated using the nltk package in Python.

In the analysis we computed GLEU and METEOR scores for every answer by BERT and DistilBERT models. Then 100 random samples were taken from the dataset and the performance of GLEU and METEOR scores were compared to our own human judgements. We scored the performance of the evaluation models based on which one gave a more accurate score compared to the other for both the results of BERT and DistilBERTs outputs. An example of manual scoring of the sample dataset is presented in Appendix 2.

There were 17 ties where both GLEU and METEOR succeeded or failed equally. Out of the 83 cases that were left, on 48 occasions (58%) the GLEU metric outperformed METEOR. This, however, was influenced by the fact that METEOR never gave a perfect 1 for a one-to-one correct answer whereas GLEU did. Crucially, METEOR failed to score 1-word responses that were exactly correct with a score of 1 and instead gave a 0. For answers that were longer than 3 words, the METEOR and GLEU scores were more comparable.

Based on evaluation results (see Appendix 4), the best compromise between the two metrics was to separate the dataset to evaluate short answers (up to 2 in length) with GLEU scores and long answers (greater than 2 in length) with METEOR scores. The short answers dataset is of length 7245 and the long answers dataset is much shorter - 3820 data points. The original data frame was split based on the lengths of the golden standard answers.

We would like to note some specifics of the dataset. In some cases, there were more than one correct answer options. For that reason, we were comparing model's answers to both gold standard options separately and only tracked the best match out of two. Data annotators were allowed to write answers in their own words, so in some cases it was not possible for the model to match the gold standard exactly and get a score of 1.

MEANS COMPARISON [A]

To evaluate whether the mean values of DistilBERT and BERT are significantly different, we conduct an independent t-test for every metric of interest: inference time, GLEU score and METEOR score.

SOFTWARE [M]

Model application, processing time tracking, and automated model evaluation were conducted in Python 3.8.3. using modules *transformers* (HuggingFace, 2020c), *PyTorch* (Paszke et al., 2019), *pandas* (McKinney, 2010), *nltk* (Bird et al., 2009), and *time* and *json* from the standard Python library.

Analysis and visualisation of processing times and evaluation metrics were conducted in R (R Core Team, 2019) using packages *tidyverse* (Wickham, 2017), *extrafont* (Winston Chang, 2014), and *rjson* (Couture-Beil, 2018).

RESULTS

MISSING ANSWERS [M]

After applying the models and receiving the outputs for BERT and DistilBERT, some of the questions were not answered by the models. The distilled model had missing answers for 446 questions, and the large model missed 303 questions. The models didn't answer questions when the highest scored answer end token in the answer prompt was found to be before the answer beginning token. The missing values were excluded from the further analysis.

For the analysis, data was selected based on the questions that were answered by both models. Since there wasn't a perfect overlap, the size of the dataset got down to 11 065 rows.

LOADING AND INFERENCE TIME [A]

Tracked loading and inference time presented in Table 8 reveal that in total DistilBERT completed the TweetQA task 529% faster than the large BERT. DistilBERT was faster in everything, except the tokenizer loading time.

Table 8. Summary of loading and inference time in ***seconds*** *on the whole dataset including non-answered questions.*

	Model loading	**Tokenizer loading**	**Inference (avg. per question)**	**Inference (total)**	**Total (loading + inference)**
DistilBERT	2.030	1.297	0.080	906.613	909.940
BERT	12.132	0.595	0.419	4804.961	4817.238

From Figure 5 (top), it's visible that the distilled model's inference times accumulate much more around it's mean and have a smaller range of values compared to the large model's values which have a long tail towards higher values. We can also see that DistilBERT made most inferences in less than 0.25 seconds with the average of 0.080 seconds, and BERT – in less than 0.75 seconds with the average of 0.429. There was one outlier in larger BERT's answers which had the processing time of 114 sec/question, which we excluded from the analysis. Using an independent t-test, we found that the observed inference time of DistilBERT (M = 0.080, SD = 0.101) was significantly lower than inference time of large BERT (M = 0.419, SD = 0.169) t(18776) = 183.91, p < .005.

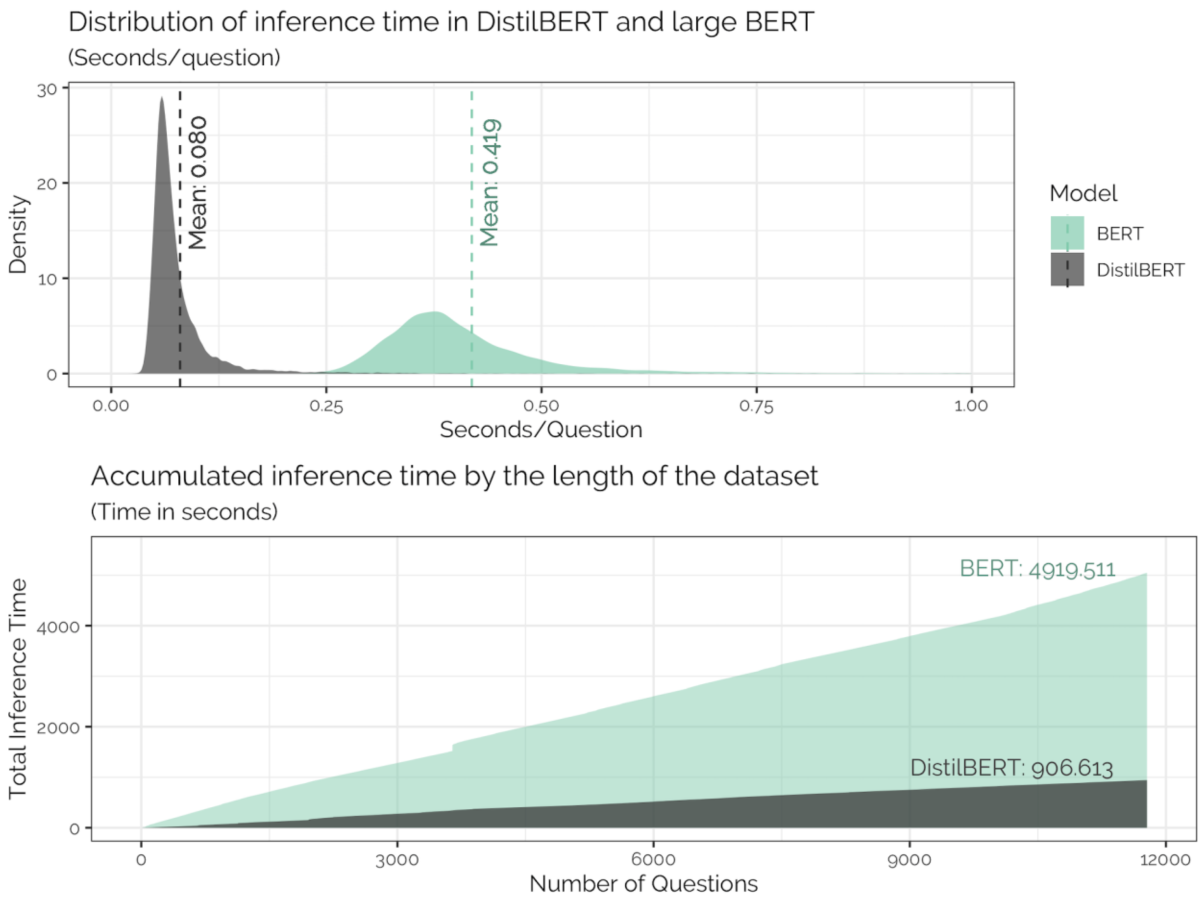


*Figure 5. **Top:** Distributions of inference time in DistilBERT and BERT (excl.118 observations that took longer than 1 sec). **Bottom:** Accumulated inference time in seconds by the number of questions in the dataset*

As can be seen in Figure 5 (bottom), along with the length of TweetQA dataset, the difference in accumulated processing time starts growing rapidly and by the end of the task exceeds an hour.

AUTOMATIC EVALUATION RESULTS [A+M]

GLEU scores presented in Figure 6 (top) were used to describe the performance on short answers which were less than 3 words in length. A score of 1 shows a one-to-one fit between the correct answer and the model's output and BERT receives more samples with such a score compared to DistilBERT. DistilBERT has more results which receive a score of 0 compared to BERT. Also, there is a larger difference between score 0 and 1 for BERT compared to DistilBERT. Interestingly, there are low GLEU scores around 0.75 and there is an extra small peak around 0.30.

Since the word lengths for GLUE scores are less than 3, we mostly expect to see the scores around 0 and 1 – either a fail or an exact match. The small peak can be explained by one word being missing, extra, or incorrect in a 2-word response. Issues with Twitter-specific data handling can also be a cause of the peculiarities in the distribution.

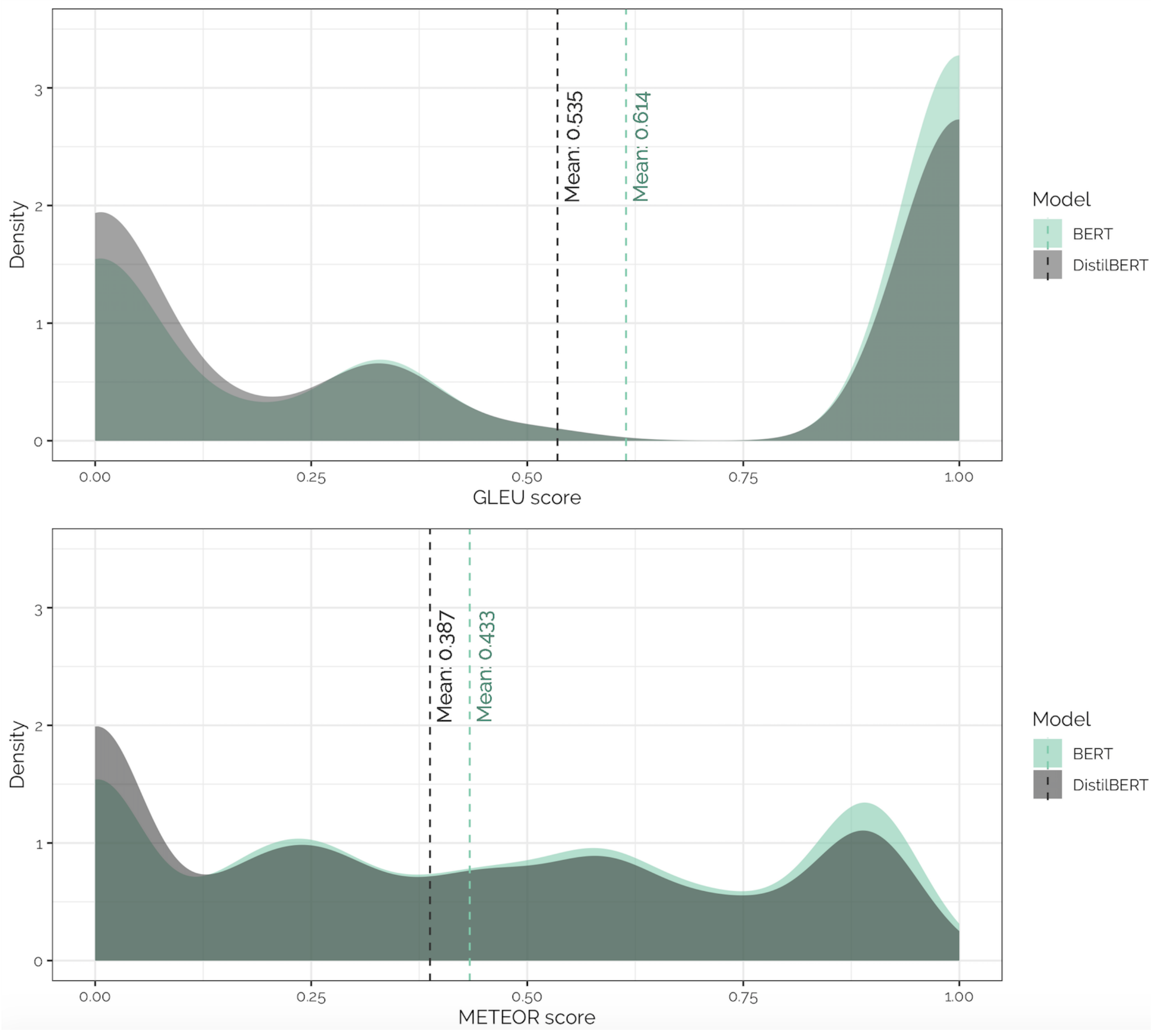


*Figure 6.**Top:** GLEU score distribution in questions with answers no longer than 2 words.*
***Bottom:** METEOR score distribution in questions with answers longer than or equal to 3 words*

METEOR scores were used to describe the performance of BERT and DistilBERT on answers that are longer than or equal to 3 words. Figure 6 (bottom) shows that there is a roughly equal distribution of METEOR scores in the limits of 0.2 and 0.8, with a peak around 0.8. BERT receives higher scores more often than DistilBERT and DistilBERT has more scores around 0.

The answers used for METEOR scores are equal to or longer than 3 words which means that there is naturally more variance in the accuracy of model outputs to the correct answers. This is why the distribution is more uniform for the different scores between 0 and 1 compared to the GLEU scores.

Table 9. General statistics of GLEU and METEOR scores for both BERT and DestilBERT models.

	GLEU		**METEOR**	
	BERT	**DistilBert**	**BERT**	**DistilBERT**
Mean	**0.61**	0.53	**0.43**	0.39
Sd	0.45	0.46	0.33	0.33
Min	0.00	0.00	0.00	0.00
25%	0.04	0.00	0.15	0.00
50%	1.00	0.33	0.43	0.35
75%	1.00	1.00	0.72	0.67
Max	1.00	1.00	1.00	1.00

Overall, the average score for BERT is higher for both models, and the means of GLEU scores are generally higher than the scores of METEOR (Table 9). As is seen from the figures as well - the standard deviations are large but very similar for both models in the evaluation metrics.

On GLEU scores, DistilBERT can replicate 87% of BERT's accuracy, and 91% on METEOR score. BERT-base and DistilBERT which were fine-tuned to SQuAD as well and performance measured with the GLUE benchmark shows that DistilBERT reaches the performance of BERT-base of around 96% (Sanh et al., 2019).

Using an independent t-test, we found that in questions with shorter answers, GLEU scores for DistilBERT (M = 0.53, SD = 0.46) were significantly lower than GLEU scores for the large BERT (M = 0.61, SD = 0.45) t(14477) = 10.526, p < .005. We similarly found that in questions with longer answers, METEOR scores for DistilBERT (M = 0.387, SD = 0.33) were also significantly lower than METEOR scores of the large BERT (M = 0.433, SD = 0.33) t(7637.8) = 6.145, p < .005.

DISCUSSION

RESULTS IN RELATION TO HYPOTHESES [A+M]

To summarize our processing time results, on average both models took less than a second to answer a question, which makes the speed difference between the models barely noticeable on a single question example. However, processing time build-up across the whole data set shows that DestilBERT is about 529% faster than BERT, and in general is more consistent in its inference time, which exceeds our expectations stated in **Hypothesis 1**. This difference can save hours of time when working with big datasets and reflects how much lower DistilBERT's computational cost is in comparison to the large BERT. Such big difference in processing time as observed in our analysis is also an indicator of much lower computational cost, suggesting that DistilBERT would be a more appropriate choice for smaller devices.

The evaluation scores indicate that BERTs results are on average closer to the golden standard by 0.08 units for GLEU scores and 0.04 units for METEOR scores, compared to DistilBERTs results. BERT also tends to output missing answers less often than DistilBERT. DistilBERT still reaches around 87-90% of large BERTs performance, which is in line with our **Hypothesis 2**. Given what both of these models can do while not being specifically fine-tuned to Twitter data, it could be possible that both of the models are ready to be used in a system where they need to retrieve information from text prompts.

Based on these results we will try to answer whether the loss in accuracy is worth the gain in processing speed when it comes to DistilBERT. For making these comparisons, it's noteworthy that we compared the large BERT to the knowledge distilled version made from the base BERT. We are comparing one of the most computationally expensive models to one of the least expensive models - usually base BERT is compared to other compressed versions.

PROCESSING SPEED VS QUALITY OF PERFORMANCE [A+M]

The most difficult question is to evaluate whether it is worth losing in speed of processing but gaining in accuracy of performance when using BERT compared to DistilBERT. Based on arbitrary margins we stated in our **Hypothesis 3**, the observed difference is processing time justifies worse performance of DistilBERT. However, we argue that the trade-off between processing cost vs evaluation success depends on where the model is applied.

For a chatbot that can answer questions given a prompt (such as if it's connected to Wikipedia), the time constraint would be to infer an answer within what is considered to be a normal time amount for answering a question. For this, BERT is probably more desirable over DistilBERT since the difference of time they take per answer is not that different, and in that case a more accurate model can be prioritized.

However, if a system needs to go through many texts to retrieve several answers quickly, such as finding the demographics information of participants in 2,000 research papers on autism for a meta-analysis, then the cumulative computational cost starts to matter. DistilBERT outperforms BERT by a large margin in that respect and with this case it might make more sense to build a DistilBERT-based model to retrieve relevant information. Another such example where DistilBERT is more advantageous would be a SoMe QA task, such as if the system is connected to Twitter API and needs to answer questions based on several tweets in real time.

These are just a few examples of QA applications, that show the nuance of choosing between the speed and the quality of performance.

LIMITATIONS AND CONSIDERATIONS

DATASET QUALITY [A]

The TweetQA dataset is both interesting and complex to use for this task. The correct answers were usually directly taken from the tweets, but sometimes they were rephrased in own words instead. Also, some but not all answers had several plausible answer options that were considered correct. This matters more for the evaluation metrics ability to score the models performance well and matters less for how the models will perform on the dataset. However, if this dataset is used for fine-tuning the model, it's unclear whether this would help make the model better or more confused at inferring answers that were not directly in the answer prompt.

Size-wise this dataset seems to be big enough for the metrics to be meaningful to compare performance between the models. For fine-tuning it might be smaller than is desired, given that a portion of it is necessary for testing.

EVALUATION IS COMPLICATED – AS WE'VE LEARNED [M]

As reviewed earlier, the metrics to compare text outputs by machines in NLP tasks is complicated and there are no good ways to date to quantify how well a machine-made text fits with a human made text.

In many papers that use both BLEU and METEOR scores to evaluate their results, they often get lower METEOR scores compared to BLEU scores, and the range at which they get the averages roughly matches the range reported in this paper (Dubey et al., 2019; Hadla et al., 2015). Even when the evaluation metrics themselves have issues in terms of validity, they are useful in showing which model is generally better than the other models. The scores shouldn't thus be taken as absolute values, such as a BLEU score of 0.5 meaning that the machine text is 50% correct compared to the reference. This is true even for the task and dataset used in our paper.

METEOR algorithm gave outputs that were larger than 1, even when we used optimized parameters, and additionally it failed to assign 1s to unigram one-to-one correct answers (when the correct answer is "blue" and both models produced "blue"). There were also some mismatches when the models' outputs said "cats" instead of "cat". In some cases, both should be considered as the right answer, but in others only one can be correct.

For these reasons we included a human evaluation process where we eyeballed the results of the evaluation metrics for both of the models and ended up splitting the data frame to two in order to get more accuracy out of the evaluation metrics. Even so, both GLEU and METEOR score still suffer from downsides and should be received with skepticism. For example, when a username was depicted written together, and the right answer wrote the name apart, neither metrics could evaluate the outputs of the model correctly.

Overall, even if the models perform better on this dataset, the metrics used to compare their results will need to be chosen with caution and approached by checking their quality via random sampling.

RECOMMENDATIONS FOR FUTURE RESEARCH [M]

Now that we have seen what the ready-made fine-tuned versions of BERT and DistilBERT can do, it would be interesting to test how much their performance would change if we were to fine-tune them on this dataset before testing them on a validation fraction of the same dataset.

It's expected that the models would handle Twitter-specific data better. They could for example distinguish usernames, hashtags, and time stamps of the tweets. Having been trained on this kind of dataset would thus already improve the quality of the already good responses that the two models are giving.

As mentioned in theory, knowledge distilled models tend to be more generalized than their teacher models. This means a few things, for example they should be able to generalize to new data better than another model that was made to be smaller than the teacher and was trained on the same dataset. It also means that a higher learning rate can be used during fine-tuning - this would possibly speed up the fine-tuning process, providing DistilBERT with even bigger time-related gains.

The pre-training and fine-tuning of BERT has already been revised previously and a new model, RoBERTa (a Robustly Optimized BERT Pretraining Approach), has been published to have better baseline training in longer batch sizes and better tuned hyperparameters. HuggingFace has also made a distilled version of RoBERTa. It's possible that the best results would be achieved by fine-tuning these versions of BERT on this dataset and using the hyperparameter suggestions reported in their paper (Liu et al., 2019). The problem with RoBERTa compared to BERT is that a large part of its improved performance is due to it being trained on more data for longer. As far as machine learning models go, it's definitely good to optimize for accuracy in performance but ideally we'd have models that have more sophisticated structures and parameter settings which would make them able to generalize on smaller amounts of data. Thus, optimizing BERT and DistilBERT is a more interesting task when we do not simply increase batch size and training data set size, but instead try to optimize their structure and hyperparameters.

When Sanh et al. (2019) added another distillation step during fine-tuning of their DistilBERT, they reached a model performance of 98% of the BERT-base, compared to 96% that they had with just one step of model distillation. This would indicate even more that focusing on improving the distilled model with knowledge distillation methods would yield improved performance.

CONCLUSION [A+M]

In this paper we compared a pre-trained and fine-tuned large BERT to DistilBERT for Question Answering. We applied both models on the TweetQA dataset, a more challenging social media dataset. We evaluated the results with adjusted GLEU and METEOR scores on an answer-level basis and found that BERT outperformed DistilBERT based on both metrics, but the difference wasn’t as large as we initially expected. In terms of processing time, DistilBERT took 529% less time than the large BERT. Even though per iteration the averages are quite similar (under 0.5 sec), in terms of cumulative costs on larger datasets DistilBERT definitely wins. Authors made a few suggestions on how this payoff between performance quality and speed could be used in different settings.

The next step is to fine-tune both models on Twitter dataset and see how much the evaluation scores change. Overall, it’s important to focus on improving the existing models via their architecture and hyperparameter settings in order to improve performance. Using knowledge distillation during both pre-training and fine-tuning of DistilBERT already improves its performance, and thus other methods of model compression could be applied to further enhance its performance, e.g. using another language model as a second teacher network.

REFERENCES

Bird, S., Loper, E., & Klein, E. (2009). *Natural Language Processing with Python* [Python]. O'Reilly Media Inc.

Buciluǎ, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 535–541.

Couture-Beil, A. (2018). *rjson: JSON for R*. https://CRAN.R-project.org/package=rjson

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv:1810.04805 [Cs]*. http://arxiv.org/abs/1810.04805

Dubey, A., Joshi, A., & Bhattacharyya, P. (2019). Deep Models for Converting Sarcastic Utterances into their Non Sarcastic Interpretation. *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data - CoDS-COMAD '19*, 289–292. https://doi.org/10.1145/3297001.3297043

Frankle, J., & Carbin, M. (2019). The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *ArXiv:1803.03635 [Cs]*. http://arxiv.org/abs/1803.03635

Gordon, M. A. (2020, January 13). *Do We Really Need Model Compression?* Mitchell A. Gordon. http://mitchgordon.me/machine/learning/2020/01/13/do-we-really-need-model-compression.html

Hadla, L., Hailat, T., & Al-Kabi, M. (2015). Comparative Study Between METEOR and BLEU Methods of MT: Arabic into English Translation as a Case Study. *International Journal of Advanced Computer Science and Applications*, *6*(11). https://doi.org/10.14569/IJACSA.2015.061128

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. *ArXiv:1503.02531 [Cs, Stat]*. http://arxiv.org/abs/1503.02531

HuggingFace. (2020a). *DistilBERT — transformers 2.10.0 documentation*. https://huggingface.co/transformers/model_doc/distilbert.html

HuggingFace. (2020b). *Loading Google AI or OpenAI pre-trained weights or PyTorch dump—Transformers 2.10.0 documentation*. https://huggingface.co/transformers/serialization.html

HuggingFace. (2020c). *Pretrained models—Transformers 2.10.0 documentation*. https://huggingface.co/transformers/pretrained_models.html

Lavie, A., & Agarwal, A. (2007). Meteor: An automatic metric for MT evaluation with high levels of correlation with human judgments. *Proceedings of the Second Workshop on Statistical Machine Translation - StatMT '07*, 228–231. https://doi.org/10.3115/1626355.1626389

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *ArXiv:1907.11692 [Cs]*. http://arxiv.org/abs/1907.11692

McCormic, C. (2020). *Question Answering with a Fine-Tuned BERT · Chris McCormick*. https://mccormickml.com/2020/03/10/question-answering-with-a-fine-tuned-BERT/

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). https://doi.org/10.25080/Majora-92bf1922-00a

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). Bleu: A Method for Automatic Evaluation of Machine Translation. *Proceedings of the 40th Annual Meeting of the*

Association for Computational Linguistics, 311–318. https://doi.org/10.3115/1073083.1073135

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

R Core Team. (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). SQuAD: 100,000+ Questions for Machine Comprehension of Text. *ArXiv:1606.05250 [Cs]*. http://arxiv.org/abs/1606.05250

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *ArXiv:1910.01108 [Cs]*. http://arxiv.org/abs/1910.01108

Tatman, R. (2019). *Evaluating Text Output in NLP: BLEU at your own risk*. https://towardsdatascience.com/evaluating-text-output-in-nlp-bleu-at-your-own-risk-e8609665a213

Tenney, I., Das, D., & Pavlick, E. (2019). BERT Rediscovers the Classical NLP Pipeline. *ArXiv:1905.05950 [Cs]*. http://arxiv.org/abs/1905.05950

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 5998–6008). Curran Associates, Inc. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Vijay, R. (2019, November 18). *Knowledge Distillation—A technique developed for compacting and accelerating Neural Nets*. Medium. https://towardsdatascience.com/knowledge-distillation-a-technique-developed-for-compacting-and-accelerating-neural-nets-732098cde690

Wickham, H. (2017). *tidyverse: Easily Install and Load the "Tidyverse."* https://CRAN.R-project.org/package=tidyverse

Winston Chang. (2014). *extrafont: Tools for using fonts*. https://CRAN.R-project.org/package=extrafont

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., … Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *ArXiv:1609.08144 [Cs]*. http://arxiv.org/abs/1609.08144

Xiong, W., Wu, J., Wang, H., Kulkarni, V., Yu, M., Chang, S., Guo, X., & Wang, W. Y. (2019). TWEETQA: A Social Media Focused Question Answering Dataset. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 5020–5031. https://doi.org/10.18653/v1/P19-1496

Zhang, T. (2014). *Geoffrey Hinton's Dark Knowledge of Machine Learning · Firstprayer*. https://firstprayer.github.io/hinton-dark-knowledge

APPENDIX 1. DATASET SUMMARY

	Original TweetQA	**Our subset of TweetQA**
# of question-answer pairs	13 757 10 692 (train) + 1086 (dev) +1979 (test)	11 778 (all used as test)
Average question length (# of words)	6.95	6.97
Average answer length (# of words in the longest answer option)	2.45	2.5

APPENDIX 2. METEOR PARAMETER VALUES

Table 10. Parameter values in the original METEOR algorithm and the adjusted values based on Lavie & Agarwal (2007)

	Original value	**Adjusted value**
α	0.90	0.81
β	3	0.83
γ	0.5	0.28

APPENDIX 3. MANUAL EVALUATION OF METRICS

Table 11. Fifteen examples from the random sample of 100 lines from the data set, the answers suggested by BERT and DistilBERT, the scores they received from METEOR and GLEU. The final column shows the judgement given by the authors of this paper – 0 shows that METEOR gives a more accurate score, and 1 means GLEU is better.

	BERT	DistilBERT	Gold_1	Gold_2	BERT_METEOR	DistilBERT_METEOR	BERT_GLEU	DistilBERT_GLEU	JUDGE
4936	his own	sean fennessey	he followed his own path.		0,380357477	0	0,214285714	0	0
3491	# restartlhc splash	restartlhc splash events	#restartlhc		0	0	0,333333333	0,166666667	1
10066	the office of the president	the president	office of the president		0,870068772	0,46546509	0,714285714	0,3	0
4760	sand bags	sand bags	sand bags		0,842491812	0,842491812	1	1	1
5200	nobel prize	nobel prize	a noble price		0	0	0	0	
2265	white hart lane	swansea city fc	white hart lane		0,88750133	0	1	0	1
3354	smiles	a 2003 in touch	she smiles		0,397790055	0	0,333333333	0	0
5832	sound check	sound check	a sound check		0,5996383	0,5996383	0,5	0,5	0
802	casting zoe saldana and darkening her skin	casting zoe saldana and darkening her skin	darkening skin	darkening zoe saldana's skin	0,488135593	0,488135593	0,136363636	0,136363636	0
7168	stopping the perpetrator	stopping the perpetrator and 2 - empowering the victims	stopping the perpetrator.		0,561661208	0,347826087	1	0,230769231	1
314	tom hiddleston	tom hiddleston	tom hiddleston	tim hiddleston	0,842491812	0,842491812	1	1	1
1249	pro - putin	pro - putin	putin.		0	0	0,333333333	0,333333333	1
2343	2022	2022	2022		0,72	0,72	1	1	1
6548	empire state building	empire state building	the empire state building.		0,44225292	0,44225292	0,6	0,6	1
3056	more affordable health care	what matters in long run is better , more affordable health care for americans	health care		0,707976313	0,393687763	0,3	0,065217391	0

APPENDIX 4. RESULTS OF METRIC EVALUATION

Gold Standard Length	GLEU correct	METEOR correct
Up to 2	**39**	21
Greater than 2	9	**13**
Up to 3	**42**	**26**
Greater than 3	6	8

APPENDIX 5. CODE

MODEL APPLICATION ON TWEETQA IN PYTHON

[https://github.com/marissala/data-science-bert/blob/master/twitterQA_bert_battle.py]

```
!pip install torch       # to access the pre-trained model weights
!pip install transformers # to access the models


# Import modules
import time
import json
import pandas as pd
import torch
from transformers import BertForQuestionAnswering
from transformers import DistilBertForQuestionAnswering
from transformers import BertTokenizer
from transformers import DistilBertTokenizer



# Import data: only dev and train since they have gold standard
# Data available at: https://tweetqa.github.io/
filenames = ["TweetQA_data/dev.json", "TweetQA_data/train.json"]

big_df = pd.DataFrame()
for file_name in filenames:
   with open(file_name, "r") as f:
       df = json.load(f)
   df = pd.DataFrame(df)
   # append data from every file to one large dataframe
   big_df = big_df.append(df, ignore_index=True)

# Load both models, track loading time, make a model list
start = time.time()
bert = BertForQuestionAnswering.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-
squad')
bert_time = time.time() - start

start = time.time()
distilbert = DistilBertForQuestionAnswering.from_pretrained('distilbert-base-uncased-distilled-
squad')
distilbert_time = time.time() - start
```

```
# Write models into a dictionary to iterate over later
model_dict = {
   "L_BERT": bert,
   "DistilBERT": distilbert
}


# Write loading times down
model_loading_times = {
   "L_BERT": str(bert_time),
   "DistilBERT": str(distilbert_time)
}


# Define QA util functions for BERT and DistilBERT
def bert_QA(row):
   # Get tweet text to fetch answer from
   answer_text = row['Tweet']

   # Get question
   question = row['Question']
   # Start tracking time
   start = time.time()

   # Apply the tokenizer to the input text, treating them as a text-pair.
   input_ids = tokenizer.encode(question, answer_text)

   # Get start and end scores for answer selection
   start_scores, end_scores = model(torch.tensor([input_ids]))

   # Find the tokens with the highest 'start' and 'end' scores.
   answer_start = torch.argmax(start_scores)
   answer_end = torch.argmax(end_scores)
   ans_tokens = input_ids[answer_start:answer_end+1]
   answer_tokens = tokenizer.convert_ids_to_tokens(ans_tokens,
                                                   skip_special_tokens=True)
   # Combine the tokens in the answer and print it out.
   row[answer_id] = tokenizer.convert_tokens_to_string(answer_tokens)

   # track time from application of tokenizer to getting the answer
   row[time_id] = str(time.time() - start)
   return row
```

```
def distil_bert_QA(row):
   # Get tweet text to fetch answer from
   answer_text = row['Tweet']

   # Get question
   question = row['Question']
   # Start tracking time
   start = time.time()

   # Apply the tokenizer to the input text, treating them as a text-pair.
   encoding = tokenizer.encode_plus(question, answer_text)
   input_ids, att_mask = encoding["input_ids"], encoding["attention_mask"]

   # Get start and end scores for answer selection
   start_scores, end_scores = model(torch.tensor([input_ids]),
                                    attention_mask=torch.tensor([att_mask]))
   ans_tokens = input_ids[torch.argmax(start_scores) : torch.argmax(end_scores) + 1]
   answer_tokens = tokenizer.convert_ids_to_tokens(ans_tokens,
                                                   skip_special_tokens=True)

   # Combine the tokens in the answer and write it down
   row[answer_id] = tokenizer.convert_tokens_to_string(answer_tokens)
   # track time from application of tokenizer to getting the answer
   row[time_id] = str(time.time() - start)
   return row

# Apply models to the data, depending on model name load appropriate tokenizer
for m_name, model in model_dict.items():
   print(f"Running {m_name}")
   answer_id = f"{m_name}_answer"
   time_id = f"{m_name}_time"
   # choose tokenizer
   if m_name == "DistilBERT":
       tok_start = time.time()
       tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased',
return_token_type_ids=True)
       distilbert_tok_time = time.time() - tok_start
       big_df = big_df.apply(distil_bert_QA, axis=1)
   else:
       tok_start = time.time()
       tokenizer = BertTokenizer.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-
squad')
       bert_tok_time = time.time() - tok_start
       # Run BERT QA on data, track time
       big_df = big_df.apply(bert_QA, axis=1)
```

```
# Write tokenizer loading times down
tokenizer_loading_times = {
   "L_BERT": str(bert_tok_time),
   "DistilBERT": str(distilbert_tok_time)
}



# Write models' inference and inference time outputs down
big_df.to_csv('twitterQA_berts.csv')

with open('model_loading.txt', 'w') as file:
   file.write(json.dumps(model_loading_times))

with open('tokenizer_loading.txt', 'w') as file:
   file.write(json.dumps(tokenizer_loading_times))
```

INFERENCE EVALUATION: USING GLEU AND METEOR SCORES IN PYTHON

[https://github.com/marissala/data-science-bert/blob/master/Evaluation.ipynb]

Evaluation

May 27, 2020

1 Evaluation of BERT and DistilBERT

1.1 METEOR and GLEU scores

1. Functions for calculating GLEU and METEOR for two outputs at the same time
2. Calculating GLEU scores
3. Calculating METEOR scores
4. Appending to dataframe, getting 50 samples

[7]:
```
import nltk
import json
import re
import numpy
import pandas as pd
import string

from nltk.translate import gleu_score
from nltk.stem.porter import PorterStemmer
from nltk.corpus import wordnet
from itertools import chain, product
```

1.2 1. Functions for calculating GLEU and METEOR for two outputs at the same time

The original GLEU function compares one translation to several golden standards, but we want to compare the performance of both BERT and DistilBERT at the same time to the golden standard.

GLEU takes an input of a split string. METEOR takes just strings.

1.2.1 1.1 GLEU functions

[8]:
```
# Function that expands the gleu sentence calculations to be done on more than␣
 ↪1 input texts
def gleu_lists_no_df(golden_standard, text1, text2):

    # n-gram lists for text1 and text2
    ngram_list = [[],[]]

    # Append the calculated gleu scores to a list for both text1 and text2
    ngram_list[0].append(nltk.translate.gleu_score.
↪sentence_gleu([golden_standard], text1))
    ngram_list[1].append(nltk.translate.gleu_score.
↪sentence_gleu([golden_standard], text2))

    # This one returns ngram_list directly for the function that makes a row␣
↪per answer
    return(ngram_list)

# Function that takes columns as input and outputs a dataframe of the results␣
↪per model
def gleu_more(gold1, gold2, berts, distilberts):
    # Initiate results dataframe
    results = pd.DataFrame()

    # Loop over the rows in the dataset
    for i in range(0,len(gold1)):
        # Define the inputs to the bleu function from the dataset
        # One row per input
        gold_1 = gold1[i]
        # Clean of punctuation
        gold_1 = gold_1.translate(str.maketrans('', '', string.punctuation))
        gold_2 = gold2[i]
        #gold_2 = gold_2.translate(str.maketrans('', '', string.punctuation))
        bert = berts[i]
        bert = bert.translate(str.maketrans('', '', string.punctuation))
        distil = distilberts[i]
        distil = distil.translate(str.maketrans('', '', string.punctuation))

        # Get the gleu scores per line in dataframe with the predefined␣
↪function for 1st golden answers
        ngram_list1 = gleu_lists_no_df(gold_1.split(), bert.split(), distil.
↪split())

        # Now get the glue scores for 2nd golden answer in case there are more␣
↪than 1 ways to answer the question
        # For missing values in Gold2, return all 0-s
        if pd.isnull(gold_2):
            ngram_list2 = [[0],[0]]
        else:
            ngram_list2 = gleu_lists_no_df(gold_2.split(), bert.split(), distil.
↪split())

        # Get the highest value per model in terms of them matching best to␣
↪gold1 or gold2
        ngram_max = [max(i, j) for i, j in zip(ngram_list1, ngram_list2)]
        # Append to dataframe, index is model name + iteration
        dff = pd.DataFrame(ngram_max, index =['0', '1'])

        # Append results to the dataframe
        results = results.append(dff)

    return(results)
```

1.2.2 1.2 METEOR functions

We are using other parameter values for meteor score calculations so I will add the whole code on compliling meteor scores together with how the parameter values were changed. Meteor score code retrieved from: https://www.nltk.org/_modules/nltk/translate/meteor_score.html

[20]:
```
from nltk.stem.porter import PorterStemmer
from nltk.corpus import wordnet
from itertools import chain, product


def _generate_enums(hypothesis, reference, preprocess=str.lower):
    """
    Takes in string inputs for hypothesis and reference and returns
    enumerated word lists for each of them

    :param hypothesis: hypothesis string
    :type hypothesis: str
    :param reference: reference string
    :type reference: str
    :preprocess: preprocessing method (default str.lower)
    :type preprocess: method
    :return: enumerated words list
    :rtype: list of 2D tuples, list of 2D tuples
    """
    hypothesis_list = list(enumerate(preprocess(hypothesis).split()))
    reference_list = list(enumerate(preprocess(reference).split()))
    return hypothesis_list, reference_list


def exact_match(hypothesis, reference):
    """
    matches exact words in hypothesis and reference
    and returns a word mapping based on the enumerated
    word id between hypothesis and reference

    :param hypothesis: hypothesis string
    :type hypothesis: str
    :param reference: reference string
    :type reference: str
    :return: enumerated matched tuples, enumerated unmatched hypothesis tuples,
             enumerated unmatched reference tuples
    :rtype: list of 2D tuples, list of 2D tuples,  list of 2D tuples
    """
    hypothesis_list, reference_list = _generate_enums(hypothesis, reference)
    return _match_enums(hypothesis_list, reference_list)



def _match_enums(enum_hypothesis_list, enum_reference_list):
    """
    matches exact words in hypothesis and reference and returns
    a word mapping between enum_hypothesis_list and enum_reference_list
    based on the enumerated word id.

    :param enum_hypothesis_list: enumerated hypothesis list
    :type enum_hypothesis_list: list of tuples
    :param enum_reference_list: enumerated reference list
    :type enum_reference_list: list of 2D tuples
    :return: enumerated matched tuples, enumerated unmatched hypothesis tuples,
             enumerated unmatched reference tuples
    :rtype: list of 2D tuples, list of 2D tuples,  list of 2D tuples
    """
    word_match = []
    for i in range(len(enum_hypothesis_list))[::-1]:
        for j in range(len(enum_reference_list))[::-1]:
            if enum_hypothesis_list[i][1] == enum_reference_list[j][1]:
                word_match.append(
                    (enum_hypothesis_list[i][0], enum_reference_list[j][0])
                )
                (enum_hypothesis_list.pop(i)[1], enum_reference_list.pop(j)[1])
                break
    return word_match, enum_hypothesis_list, enum_reference_list


def _enum_stem_match(
    enum_hypothesis_list, enum_reference_list, stemmer=PorterStemmer()
):
    """
    Stems each word and matches them in hypothesis and reference
    and returns a word mapping between enum_hypothesis_list and
    enum_reference_list based on the enumerated word id. The function also
    returns a enumerated list of unmatched words for hypothesis and reference.

    :param enum_hypothesis_list:
```

```
    :type enum_hypothesis_list:
    :param enum_reference_list:
    :type enum_reference_list:
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem␣
↪method
    :return: enumerated matched tuples, enumerated unmatched hypothesis tuples,
             enumerated unmatched reference tuples
    :rtype: list of 2D tuples, list of 2D tuples,  list of 2D tuples
    """
    stemmed_enum_list1 = [
        (word_pair[0], stemmer.stem(word_pair[1])) for word_pair in␣
↪enum_hypothesis_list
    ]

    stemmed_enum_list2 = [
        (word_pair[0], stemmer.stem(word_pair[1])) for word_pair in␣
↪enum_reference_list
    ]

    word_match, enum_unmat_hypo_list, enum_unmat_ref_list = _match_enums(
        stemmed_enum_list1, stemmed_enum_list2
    )

    enum_unmat_hypo_list = (
        list(zip(*enum_unmat_hypo_list)) if len(enum_unmat_hypo_list) > 0 else␣
↪[]
    )

    enum_unmat_ref_list = (
        list(zip(*enum_unmat_ref_list)) if len(enum_unmat_ref_list) > 0 else []
    )

    enum_hypothesis_list = list(
        filter(lambda x: x[0] not in enum_unmat_hypo_list, enum_hypothesis_list)
    )

    enum_reference_list = list(
        filter(lambda x: x[0] not in enum_unmat_ref_list, enum_reference_list)
    )

    return word_match, enum_hypothesis_list, enum_reference_list


def stem_match(hypothesis, reference, stemmer=PorterStemmer()):
    """
    Stems each word and matches them in hypothesis and reference
    and returns a word mapping between hypothesis and reference

    :param hypothesis:
    :type hypothesis:
    :param reference:
    :type reference:
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that
                   implements a stem method
    :return: enumerated matched tuples, enumerated unmatched hypothesis tuples,
             enumerated unmatched reference tuples
    :rtype: list of 2D tuples, list of 2D tuples,  list of 2D tuples
    """
    enum_hypothesis_list, enum_reference_list = _generate_enums(hypothesis,␣
↪reference)
    return _enum_stem_match(enum_hypothesis_list, enum_reference_list,␣
↪stemmer=stemmer)



def _enum_wordnetsyn_match(enum_hypothesis_list, enum_reference_list,␣
↪wordnet=wordnet):
    """
    Matches each word in reference to a word in hypothesis
    if any synonym of a hypothesis word is the exact match
    to the reference word.

    :param enum_hypothesis_list: enumerated hypothesis list
    :param enum_reference_list: enumerated reference list
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :return: list of matched tuples, unmatched hypothesis list, unmatched␣
↪reference list
    :rtype:  list of tuples, list of tuples, list of tuples

    """
    word_match = []
    for i in range(len(enum_hypothesis_list))[::-1]:
        hypothesis_syns = set(
            chain(
                *[
                    [
                        lemma.name()
                        for lemma in synset.lemmas()
                        if lemma.name().find("_") < 0
                    ]
                    for synset in wordnet.synsets(enum_hypothesis_list[i][1])
                ]
            )
        ).union({enum_hypothesis_list[i][1]})
        for j in range(len(enum_reference_list))[::-1]:
            if enum_reference_list[j][1] in hypothesis_syns:
                word_match.append(
                    (enum_hypothesis_list[i][0], enum_reference_list[j][0])
                )
                enum_hypothesis_list.pop(i), enum_reference_list.pop(j)
                break
    return word_match, enum_hypothesis_list, enum_reference_list


def wordnetsyn_match(hypothesis, reference, wordnet=wordnet):
    """
    Matches each word in reference to a word in hypothesis if any synonym
    of a hypothesis word is the exact match to the reference word.

    :param hypothesis: hypothesis string
    :param reference: reference string
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :return: list of mapped tuples
    :rtype: list of tuples
    """
    enum_hypothesis_list, enum_reference_list = _generate_enums(hypothesis,␣
 ↪reference)
    return _enum_wordnetsyn_match(
        enum_hypothesis_list, enum_reference_list, wordnet=wordnet
    )



def _enum_allign_words(
    enum_hypothesis_list, enum_reference_list, stemmer=PorterStemmer(),␣
 ↪wordnet=wordnet
):
    """
    Aligns/matches words in the hypothesis to reference by sequentially
    applying exact match, stemmed match and wordnet based synonym match.
    in case there are multiple matches the match which has the least number
    of crossing is chosen. Takes enumerated list as input instead of
    string input

    :param enum_hypothesis_list: enumerated hypothesis list
    :param enum_reference_list: enumerated reference list
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem␣
↪method
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :return: sorted list of matched tuples, unmatched hypothesis list,
             unmatched reference list
    :rtype: list of tuples, list of tuples, list of tuples
    """
    exact_matches, enum_hypothesis_list, enum_reference_list = _match_enums(
        enum_hypothesis_list, enum_reference_list
    )

    stem_matches, enum_hypothesis_list, enum_reference_list = _enum_stem_match(
        enum_hypothesis_list, enum_reference_list, stemmer=stemmer
    )

    wns_matches, enum_hypothesis_list, enum_reference_list =␣
↪_enum_wordnetsyn_match(
        enum_hypothesis_list, enum_reference_list, wordnet=wordnet
    )

    return (
        sorted(
            exact_matches + stem_matches + wns_matches, key=lambda wordpair:␣
↪wordpair[0]
        ),
        enum_hypothesis_list,
        enum_reference_list,
    )


def allign_words(hypothesis, reference, stemmer=PorterStemmer(),␣
↪wordnet=wordnet):
    """
    Aligns/matches words in the hypothesis to reference by sequentially
    applying exact match, stemmed match and wordnet based synonym match.
    In case there are multiple matches the match which has the least number
    of crossing is chosen.

    :param hypothesis: hypothesis string
    :param reference: reference string
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem␣
↪method
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :return: sorted list of matched tuples, unmatched hypothesis list,␣
↪unmatched reference list
    :rtype: list of tuples, list of tuples, list of tuples
    """
    enum_hypothesis_list, enum_reference_list = _generate_enums(hypothesis,␣
↪reference)
    return _enum_allign_words(
        enum_hypothesis_list, enum_reference_list, stemmer=stemmer,␣
↪wordnet=wordnet
    )



def _count_chunks(matches):
    """
    Counts the fewest possible number of chunks such that matched unigrams
    of each chunk are adjacent to each other. This is used to caluclate the
    fragmentation part of the metric.

    :param matches: list containing a mapping of matched words (output of␣
↪allign_words)
    :return: Number of chunks a sentence is divided into post allignment
    :rtype: int
    """
    i = 0
    chunks = 1
    while i < len(matches) - 1:
        if (matches[i + 1][0] == matches[i][0] + 1) and (
            matches[i + 1][1] == matches[i][1] + 1
        ):
            i += 1
            continue
        i += 1
        chunks += 1
    return chunks


def single_meteor_score(
    reference,
    hypothesis,
    preprocess=str.lower,
    stemmer=PorterStemmer(),
    wordnet=wordnet,
    alpha=0.9,
    beta=3,
    gamma=0.5,
):
    """
    Calculates METEOR score for single hypothesis and reference as per
    "Meteor: An Automatic Metric for MT Evaluation with HighLevels of
    Correlation with Human Judgments" by Alon Lavie and Abhaya Agarwal,
    in Proceedings of ACL.
    http://www.cs.cmu.edu/~alavie/METEOR/pdf/Lavie-Agarwal-2007-METEOR.pdf


    >>> hypothesis1 = 'It is a guide to action which ensures that the military␣
↪always obeys the commands of the party'

    >>> reference1 = 'It is a guide to action that ensures that the military␣
↪will forever heed Party commands'


    >>> round(single_meteor_score(reference1, hypothesis1),4)
    0.7398

        If there is no words match during the alignment the method returns the
        score as 0. We can safely  return a zero instead of raising a
        division by zero error as no match usually implies a bad translation.

    >>> round(meteor_score('this is a cat', 'non matching hypothesis'),4)
    0.0

    :param references: reference sentences
    :type references: list(str)
    :param hypothesis: a hypothesis sentence
    :type hypothesis: str
    :param preprocess: preprocessing function (default str.lower)
    :type preprocess: method
    :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
    :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem␣
↪method
    :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
    :type wordnet: WordNetCorpusReader
    :param alpha: parameter for controlling relative weights of precision and␣
↪recall.
    :type alpha: float
    :param beta: parameter for controlling shape of penalty as a
                 function of as a function of fragmentation.
    :type beta: float
    :param gamma: relative weight assigned to fragmentation penality.
    :type gamma: float
    :return: The sentence-level METEOR score.
    :rtype: float
    """
    enum_hypothesis, enum_reference = _generate_enums(
        hypothesis, reference, preprocess=preprocess
    )
    translation_length = len(enum_hypothesis)
    reference_length = len(enum_reference)
    matches, _, _ = _enum_allign_words(enum_hypothesis, enum_reference,␣
↪stemmer=stemmer)
    matches_count = len(matches)
    try:
        precision = float(matches_count) / translation_length
        recall = float(matches_count) / reference_length
        fmean = (precision * recall) / (alpha * precision + (1 - alpha) *␣
↪recall)
        chunk_count = float(_count_chunks(matches))
        frag_frac = chunk_count / matches_count
    except ZeroDivisionError:
        return 0.0
    penalty = gamma * frag_frac ** beta
    return (1 - penalty) * fmean



def meteor_score(
    references,
    hypothesis,
    preprocess=str.lower,
    stemmer=PorterStemmer(),
    wordnet=wordnet,
    # Original parameter values:
    #alpha=0.9,
    #beta=3,
    #gamma=0.5,
    # Parameter values we chose based on the paper: https://www.cs.cmu.edu/
↪~alavie/METEOR/pdf/Lavie-Agarwal-2007-METEOR.pdf
    alpha=0.81,
    beta=0.83,
    gamma=0.28,
):
    """
    Calculates METEOR score for hypothesis with multiple references as
    described in "Meteor: An Automatic Metric for MT Evaluation with
    HighLevels of Correlation with Human Judgments" by Alon Lavie and
    Abhaya Agarwal, in Proceedings of ACL.
    http://www.cs.cmu.edu/~alavie/METEOR/pdf/Lavie-Agarwal-2007-METEOR.pdf


    In case of multiple references the best score is chosen. This method
```

```
   iterates over single_meteor_score and picks the best pair among all
   the references for a given hypothesis

   >>> hypothesis1 = 'It is a guide to action which ensures that the military␣
↪always obeys the commands of the party'
   >>> hypothesis2 = 'It is to insure the troops forever hearing the activity␣
↪guidebook that party direct'

   >>> reference1 = 'It is a guide to action that ensures that the military␣
↪will forever heed Party commands'
   >>> reference2 = 'It is the guiding principle which guarantees the military␣
↪forces always being under the command of the Party'
   >>> reference3 = 'It is the practical guide for the army always to heed the␣
↪directions of the party'

   >>> round(meteor_score([reference1, reference2, reference3], hypothesis1),4)
   0.7398

       If there is no words match during the alignment the method returns the
       score as 0. We can safely  return a zero instead of raising a
       division by zero error as no match usually implies a bad translation.

   >>> round(meteor_score(['this is a cat'], 'non matching hypothesis'),4)
   0.0

   :param references: reference sentences
   :type references: list(str)
   :param hypothesis: a hypothesis sentence
   :type hypothesis: str
   :param preprocess: preprocessing function (default str.lower)
   :type preprocess: method
   :param stemmer: nltk.stem.api.StemmerI object (default PorterStemmer())
   :type stemmer: nltk.stem.api.StemmerI or any class that implements a stem␣
↪method
   :param wordnet: a wordnet corpus reader object (default nltk.corpus.wordnet)
   :type wordnet: WordNetCorpusReader
   :param alpha: parameter for controlling relative weights of precision and␣
↪recall.
   :type alpha: float
   :param beta: parameter for controlling shape of penalty as a function
                of as a function of fragmentation.
   :type beta: float
   :param gamma: relative weight assigned to fragmentation penality.
   :type gamma: float
   :return: The sentence-level METEOR score.
   :rtype: float
```

```
    """
    return max(
        [
            single_meteor_score(
                reference,
                hypothesis,
                stemmer=stemmer,
                wordnet=wordnet,
                alpha=alpha,
                beta=beta,
                gamma=gamma,
            )
            for reference in references
        ]
    )
```

[21]:

```
# Functiones defined for calculating the meteor scores

# Define the function to calculate BLEU scores for more than one inputs
def meteor_lists_no_df(golden_standard, text1, text2):
    # n-gram lists for text1 and text2
    meteor_list = [[],[]]

    # Append meteor scores - call the meteor function on text1 and text2
    meteor_list[0].append(meteor_score([golden_standard], text1))
    meteor_list[1].append(meteor_score([golden_standard], text2))

    # This one returns ngram_list directly for the function that makes a row␣
 ↪per answer
    return(meteor_list)

def meteor_more(gold1, berts, distilberts):
    # Initiate results dataframe
    results = pd.DataFrame()

    # Loop over rows in the dataframe
    for i in range(0,len(gold1)):

        # Define the inputs to the bleu function from the dataset
        gold_1 = gold1[i]
        bert = berts[i]
        distil = distilberts[i]

        # Get the bleu scores per line in dataframe with the predefined function
        meteor_list1 = meteor_lists_no_df(gold_1, bert, distil)

        # Append to dataframe, index is model name + iteration
        dff = pd.DataFrame(meteor_list1, index =['0', '1'], columns =␣
↪["METEOR"])

        # Append results to the dataframe
        results = results.append(dff)

    return(results)
```

[22]:
```
meteor_more(df['Gold_1'], df['BERT'], df['DistilBERT'])
```

[22]:
```
      METEOR
0   0.842492
1   0.842492
0   0.000000
1   0.000000
0   0.000000
..       …
1   0.000000
0   0.720000
1   0.720000
0   0.328767
1   0.328767

[22130 rows x 1 columns]
```

1.3 2. Calculate GLEU scores

1.3.1 2.1 Prepare the dataframe

[58]:
```
import pandas as pd
# Read in data
data = pd.read_csv("../tweetQA_bothpresent.csv")
```

[59]:
```
# Select columns relevant
df = data[['Answer', "L_BERT_answer", "DistilBERT_answer"]]
df = df.rename(columns={'Answer': 'Answers', "L_BERT_answer": "BERT",␣
↪"DistilBERT_answer": "DistilBERT"})
```

[60]:
```
df
```

[60]:
```
                                        Answers  \
0                              ['w nj', 'w nj']
1       ['#endangeredriver', '#endangereddriver']
2                        ['wiggins', 'wiggins']
3          ['the game is tied at 106', '106-106']
4                   ["kemba's", "kemba's floater"]
…                                              …
```

```
11060                                        ['guns']
11061                              ['president obama']
11062                       ['our best, whole foods']
11063                                     ['january']
11064                                     ['shed it']

                                                BERT  \
0                                               w nj
1                                  # endangeredriver
2       monstars basketball @ m0nstarsbballwiggins
3                                          106 - 106
4                                              kemba
…                                                  …
11060                                           guns
11061                                president obama
11062                      it happens to the best of us
11063                                        january
11064                                 shed their skin

                                          DistilBERT
0                                               w nj
1                                           jdsutter
2       monstars basketball @ m0nstarsbballwiggins
3                                  106 - 106 . 8 . 9
4                                              kemba
…                                                  …
11060                                           guns
11061                                president obama
11062                      it happens to the best of us
11063                                        january
11064                                 shed their skin

[11065 rows x 3 columns]
```

[61]:
```
df.describe()
```

[61]:
```
                  Answers   BERT         DistilBERT
count               11065  11065              11065
unique               9432   8875               8991
top     ['donald trump']  trump  donald j . trump
freq                   33     30                 26
```

The answers have sometimes more than one correct option: make the answers into two columns, Gold_1 and Gold_2.

[62]:
```
import ast
```

```
# Function to split the rows
def split_column(row):
    # Cuts the answers row into two if possible
    initial_gold = ast.literal_eval(row['Answers'])
    # Gold_1 will always have an input
    row['Gold_1'] = initial_gold[0]
    # If the list has more than 1 element, Gold_2 gets the second input
    if len(initial_gold) > 1:
        row['Gold_2'] = initial_gold[1]

    return(row)

# Apply on the dataframe
df = df.apply(split_column, axis = 1)
```

1.3.2 2.2 Apply the GLEU function

[27]:
```
import math
gleus = gleu_more(df['Gold_1'], df['Gold_2'], df['BERT'], df['DistilBERT'])
```

[28]:
```
#Define the bert and distilbert results based on the indexes given to them in␣
↪the function above
bert_results = gleus.loc["0"]
distil_results = gleus.loc["1"]
```

[97]:
```
# inspect
bert_results.describe()
```

[97]:
```
                  0
count  11065.000000
mean       0.542009
std        0.436460
min        0.000000
25%        0.055556
50%        0.500000
75%        1.000000
max        1.000000
```

[98]:
```
distil_results.describe()
```

[98]:
```
                  0
count  11065.000000
mean       0.472871
std        0.439373
min        0.000000
25%        0.000000
50%        0.333333
```

```
75%          1.000000
max          1.000000
```

[345]:
```
bert_results.to_csv("bert_gleus.csv", index = False)
distil_results.to_csv("distil_gleus.csv", index = False)
```

1.4 3. Calculate METEOR scores

[63]:
```
meteors = meteor_more(df['Gold_1'], df['BERT'], df['DistilBERT'])
```

[67]:
```
# Get the scores for bert and distilbert
bert_meteors = meteors.loc["0"]
distil_meteors = meteors.loc["1"]
```

[81]:
```
bert_meteors.describe()
```

[81]:
```
             METEOR
count  11065.000000
mean       0.460381
std        0.337903
min        0.000000
25%        0.000000
50%        0.547954
75%        0.720000
max        1.000000
```

The METEOR scores are sometimes above 1. Inspection below in part 4 shows that the score being one usually means the model was correct. Thus, the scores above 1 will be replaced with a score of 1.

[89]:
```
bert_meteors['METEOR'] = np.where(bert_meteors['METEOR'] > 1, 1,␣
 ↪bert_meteors['METEOR'])
bert_meteors.describe()
```

[89]:
```
             METEOR
count  11065.000000
mean       0.460381
std        0.337903
min        0.000000
25%        0.000000
50%        0.547954
75%        0.720000
max        1.000000
```

[90]:
```
distil_meteors['METEOR'] = np.where(distil_meteors['METEOR'] > 1, 1,␣
 ↪distil_meteors['METEOR'])
distil_meteors.describe()
```

```
[90]:                 METEOR
      count  11065.000000
      mean       0.415755
      std        0.343811
      min        0.000000
      25%        0.000000
      50%        0.397790
      75%        0.720000
      max        1.000000
```

```
[83]: bert_meteors.to_csv("bert_meteors.csv", index = False)
      distil_meteors.to_csv("distil_meteors.csv", index = False)
```

1.5 4. Appending to dataframe, getting 50 samples

```
[35]: df
```

```
[35]:                                               Answers  \
      0                                    ['w nj', 'w nj']
      1       ['#endangeredriver', '#endangereddriver']
      2                              ['wiggins', 'wiggins']
      3          ['the game is tied at 106', '106-106']
      4                      ["kemba's", "kemba's floater"]
      …                                                   …
      11060                                       ['guns']
      11061                            ['president obama']
      11062                      ['our best, whole foods']
      11063                                    ['january']
      11064                                    ['shed it']

                                                   BERT  \
      0                                            w nj
      1                               # endangeredriver
      2       monstars basketball @ m0nstarsbballwiggins
      3                                       106 - 106
      4                                           kemba
      …                                               …
      11060                                        guns
      11061                             president obama
      11062                  it happens to the best of us
      11063                                     january
      11064                             shed their skin

                                              DistilBERT                  Gold_1  \
      0                                            w nj                    w nj
      1                                        jdsutter        #endangeredriver
      2       monstars basketball @ m0nstarsbballwiggins                 wiggins
```

```
3                              106 - 106 . 8 . 9  the game is tied at 106
4                                          kemba                  kemba's
…                                              …                        …
11060                                       guns                     guns
11061                             president obama          president obama
11062                 it happens to the best of us    our best, whole foods
11063                                    january                  january
11064                             shed their skin                   shed it

                   Gold_2
0                    w nj
1      #endangereddriver
2                 wiggins
3                 106-106
4         kemba's floater
…                       …
11060                 NaN
11061                 NaN
11062                 NaN
11063                 NaN
11064                 NaN

[11065 rows x 5 columns]
```

[84]:
```
# Reset indexes for results
bert_meteors = bert_meteors.reset_index(drop=True)
distil_meteors = distil_meteors.reset_index(drop=True)
bert_results = bert_results.reset_index(drop=True)
distil_results = distil_results.reset_index(drop=True)

# Append to dataframe
df['BERT_METEOR'] = bert_meteors['METEOR']
df['DistilBERT_METEOR'] = distil_meteors['METEOR']
df['BERT_GLEU'] = bert_results[0]
df['DistilBERT_GLEU'] = distil_results[0]
```

[85]:
```
df
```

[85]:
```
                                          Answers  \
0                                ['w nj', 'w nj']
1      ['#endangereddriver', '#endangereddriver']
2                          ['wiggins', 'wiggins']
3          ['the game is tied at 106', '106-106']
4                   ["kemba's", "kemba's floater"]
…                                               …
11060                                    ['guns']
11061                         ['president obama']
```

```
11062                 ['our best, whole foods']
11063                              ['january']
11064                              ['shed it']

                                          BERT  \
0                                          w nj
1                             # endangeredriver
2      monstars basketball @ m0nstarsbballwiggins
3                                     106 - 106
4                                         kemba
…                                            …
11060                                      guns
11061                           president obama
11062                  it happens to the best of us
11063                                   january
11064                           shed their skin

                                    DistilBERT                 Gold_1  \
0                                          w nj                  w nj
1                                      jdsutter      #endangeredriver
2      monstars basketball @ m0nstarsbballwiggins              wiggins
3                              106 - 106 . 8 . 9  the game is tied at 106
4                                         kemba               kemba's
…                                            …                    …
11060                                      guns                  guns
11061                           president obama       president obama
11062                  it happens to the best of us   our best, whole foods
11063                                   january               january
11064                           shed their skin               shed it

                 Gold_2  BERT_METEOR  DistilBERT_METEOR  BERT_GLEU  \
0                  w nj     0.842492           0.842492   1.000000
1      #endangereddriver     0.000000           0.000000   1.000000
2               wiggins     0.000000           0.000000   0.000000
3               106-106     0.132597           0.116317   0.055556
4         kemba's floater     0.000000           0.000000   0.000000
…                     …          …                  …          …
11060                NaN     0.720000           0.720000   1.000000
11061                NaN     0.842492           0.842492   1.000000
11062                NaN     0.000000           0.000000   0.045455
11063                NaN     0.720000           0.720000   1.000000
11064                NaN     0.328767           0.328767   0.166667

       DistilBERT_GLEU
0             1.000000
1             0.000000
2             0.000000
```

```
3            0.055556
4            0.000000
…               …
11060        1.000000
11061        1.000000
11062        0.045455
11063        1.000000
11064        0.166667

[11065 rows x 9 columns]
```

[102]:
```python
# Random sample 50 sentences
samples = df.sample(n=100)

# Read out samples to inspect them in Excel
samples.to_csv("df_samples_scores100.csv")
```

[87]:
```python
# Make the two dataframes for short and long answers
import numpy as np

short_answers = df.loc[np.array(list(map(len,df["Gold_1"].str.split()))) < 3]
long_answers = df.loc[np.array(list(map(len,df["Gold_1"].str.split()))) >= 3]
```

[88]:
```python
short_answers.to_csv("df_short_answers.csv")
long_answers.to_csv("df_long_answers.csv")
```

[52]:
```python
# Inspecting the dataframe where meteor scores were bigger than 1
df[df["BERT_METEOR"] => 1]
```

[52]:
```
                                  Answers                            BERT  \
449                     ['shark', 'sharks']                          sharks
1027                      ['steal from us']                 stealing from us
1099                        ['paul walker']                    paul walkers
1418                      ['cairnes store']                   cairns stores
1581   ['hundred of thousands of dollars']  hundreds of thousands of dollars
2365                              ['smile']                          smiled
2422           ['transgendered people']              transgender people
3731                   ['play racist call']              plays racist calls
4320                         ['stereotype']                     stereotypes
4769                   ['work really hard']              works really hard
5068                  ['female action star']             female action stars
5332                     ['stop to listen']              stopping to listen
5765                                ['end']                           ended
7066                                ['gun']                            guns
7373                  ['42 degree celsius']              42 degrees celsius
7936                 ['help deliver babies']           helps delivers babies
8102                 ['black makeup artist']            black makeup artists
```

```
8362                 ['dodge bullets']                  dodged bullets
8611                   ['paul walker']                    paul walkers
9685                    ['killing me']               trying to kill me
10666                   ['masturbate']              masturbated at me
10922                      ['pull it']                      pulling it

                                      DistilBERT  \
449    jason demers ( @ jasondemers5 ) july 16 , 2014
1027                                        stealing
1099                                    paul walkers
1418                                          cairns
1581                  hundreds of thousands of dollars
2365                                          smiled
2422                              transgender people
3731                              plays racist calls
4320                                     stereotypes
4769                               works really hard
5068                              female action stars
5332                               stopping to listen
5765                                           ended
7066                               kim kardashian west
7373                              42 degrees celsius
7936                            helps delivers babies
8102                             black makeup artists
8362                                  dodged bullets
8611                                      kevin hart
9685                               trying to kill me
10666                                    masturbated
10922                                        pulling

                                 Gold_1  Gold_2  BERT_METEOR  \
449                               shark  sharks     1.440000
1027                      steal from us     NaN     1.123322
1099                        paul walker     NaN     1.200019
1418                       cairnes store     NaN     1.200019
1581   hundred of thousands of dollars     NaN     1.065002
2365                              smile     NaN     1.440000
2422                transgendered people     NaN     1.200019
3731                    play racist call     NaN     1.361264
4320                          stereotype     NaN     1.440000
4769                    work really hard     NaN     1.123322
5068                   female action star     NaN     1.123322
5332                      stop to listen     NaN     1.123322
5765                                end     NaN     1.440000
7066                                gun     NaN     1.440000
7373                   42 degree celsius     NaN     1.123322
7936                  help deliver babies     NaN     1.361264
```

```
8102                black makeup artist     NaN     1.123322
8362                      dodge bullets     NaN     1.200019
8611                        paul walker     NaN     1.200019
9685                         killing me     NaN     1.008419
10666                        masturbate     NaN     1.043478
10922                           pull it     NaN     1.200019

       DistilBERT_METEOR  BERT_GLEU  DistilBERT_GLEU
449             0.000000   1.000000         0.000000
1027            0.549618   0.500000         0.000000
1099            1.200019   0.333333         0.333333
1418            0.397790   0.000000         0.000000
1581            1.065002   0.714286         0.714286
2365            1.440000   0.000000         0.000000
2422            1.200019   0.333333         0.333333
3731            1.361264   0.166667         0.166667
4320            1.440000   0.000000         0.000000
4769            1.123322   0.500000         0.500000
5068            1.123322   0.500000         0.500000
5332            1.123322   0.500000         0.500000
5765            1.440000   0.000000         0.000000
7066            0.000000   0.000000         0.000000
7373            1.123322   0.333333         0.333333
7936            1.361264   0.166667         0.166667
8102            1.123322   0.500000         0.500000
8362            1.200019   0.333333         0.333333
8611            0.000000   0.333333         0.000000
9685            1.008419   0.100000         0.100000
10666           1.440000   0.000000         0.000000
10922           0.795580   0.333333         0.000000
```

[99]:
```
short_answers.describe()
```

[99]:
```
       BERT_METEOR  DistilBERT_METEOR    BERT_GLEU  DistilBERT_GLEU
count  7245.000000        7245.000000  7245.000000      7245.000000
mean      0.474589           0.430672     0.614320         0.534981
std       0.343093           0.351143     0.447432         0.459766
min       0.000000           0.000000     0.000000         0.000000
25%       0.000000           0.000000     0.043478         0.000000
50%       0.657534           0.521739     1.000000         0.333333
75%       0.720000           0.720000     1.000000         1.000000
max       1.000000           1.000000     1.000000         1.000000
```

[100]:
```
long_answers.describe()
```

[100]:
```
       BERT_METEOR  DistilBERT_METEOR    BERT_GLEU  DistilBERT_GLEU
count  3820.000000        3820.000000  3820.000000      3820.000000
```

```
mean     0.433435           0.387465    0.404865         0.355074
std      0.326184           0.327642    0.378705         0.370297
min      0.000000           0.000000    0.000000         0.000000
25%      0.146568           0.000000    0.071429         0.000000
50%      0.426683           0.350308    0.300000         0.200000
75%      0.724701           0.665626    0.714286         0.600000
max      1.000000           1.000000    1.000000         1.000000
```

1.6 Additional code: how GLEU compares to METEOR in short examples

[39]:
```
# Meteor is lower for an exact short match
print(meteor_score(["one"], "one"))
print(nltk.translate.gleu_score.sentence_gleu(["one"], "one"))
```

```
0.72
1.0
```

[40]:
```
# Meteor is higher for when there are more words in exact match
a = "one and seventeen"
print(meteor_score([a], a))
print(nltk.translate.gleu_score.sentence_gleu([a.split()], a.split()))
```

```
0.8875013295249914
1.0
```

Performance visualisations

Anita Kurm

5/21/2020

Set-up, data import

```
pacman::p_load(tidyverse, rjson, extrafont)
font_import(prompt = FALSE, pattern = "Raleway") #you need to download the font for that
```

```
## Scanning ttf files in /Library/Fonts/, /System/Library/Fonts, ~/Library/Fonts/ ...
```

```
## Extracting .afm files from .ttf files...
```

```
## /Users/anitakurm/Library/Fonts/Raleway-Regular.ttf : Raleway-Regular already registered in fonts database
. Skipping.
## /Users/anitakurm/Library/Fonts/Raleway-Thin.ttf : Raleway-Thin already registered in fonts database. Skip
ping.
## Found FontName for 0 fonts.
## Scanning afm files in /Library/Frameworks/R.framework/Versions/3.6/Resources/library/extrafontdb/metrics
```

```
## Warning in grepl("^FamilyName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FontName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FullName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^Weight", text): input string 4 is invalid in this locale
```

```
## Warning in grepl("^FamilyName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FontName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FullName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^Weight", text): input string 4 is invalid in this locale
```

```
## Warning in grepl("^FamilyName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FontName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FullName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^Weight", text): input string 4 is invalid in this locale
```

```
## Warning in grepl("^FamilyName", text): input string 4 is invalid in this
## locale
## Warning in grepl("^FullName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^Weight", text): input string 4 is invalid in this locale
```

```
## Warning in grepl("^FamilyName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FontName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FullName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^Weight", text): input string 4 is invalid in this locale
```

```
## Warning in grepl("^FamilyName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FontName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FullName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^Weight", text): input string 4 is invalid in this locale
```

```
## Warning in grepl("^FamilyName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FontName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FullName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^Weight", text): input string 4 is invalid in this locale
```

```
## Warning in grepl("^FamilyName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FontName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^FullName", text): input string 4 is invalid in this
## locale
```

```
## Warning in grepl("^Weight", text): input string 4 is invalid in this locale
```

```r
df <- read_csv("twitterQA_berts.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   Question = col_character(),
##   Answer = col_character(),
##   Tweet = col_character(),
##   qid = col_character(),
##   L_BERT_answer = col_character(),
##   L_BERT_time = col_double(),
##   DistilBERT_answer = col_character(),
##   DistilBERT_time = col_double()
## )
```

```
tok_times <- fromJSON(file = "tokenizer_loading.txt")
mod_times <- fromJSON(file = "model_loading.txt")

# Define color palette
cp <- c("aquamarine3", "grey19")

# See available fonts
#fonts()
```

Initial dataset stats

```
init <- read_csv("initial11778.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   Answer = col_character(),
##   Gold_1 = col_character(),
##   Gold_2 = col_character(),
##   Question = col_character(),
##   Tweet = col_character(),
##   qid = col_character()
## )
```

```
init %>%
  mutate(Gold1_length = sapply(strsplit(Gold_1, " "), length),
         Gold2_length = ifelse(is.na(Gold_2), 0, sapply(strsplit(Gold_2, " "), length)),
         Gold_max_length = ifelse(Gold1_length>Gold2_length, Gold1_length, Gold2_length),
         Question_length = sapply(strsplit(Question, " "), length)) %>%
  summarise(n(), mean(Gold_max_length), mean(Question_length))
```

```
## # A tibble: 1 x 3
##   `n()` `mean(Gold_max_length)` `mean(Question_length)`
##   <int>                   <dbl>                   <dbl>
## 1 11778                    2.50                    6.97
```

Data pre-processing

```
# Reshape data by creating two separate dfs and binding together
l_bert <- df %>%
  select(qid,
         X1,
         'Answer_pred' = L_BERT_answer,
         'Time' = L_BERT_time) %>%
  mutate(A_len = str_length(Answer_pred),
         Model = "BERT",
         Tok_time = tok_times$L_BERT,
         Load_time = mod_times$L_BERT)

d_bert <- df %>%
  select(qid,
         X1,
         'Answer_pred' = DistilBERT_answer,
         'Time' = DistilBERT_time) %>%
  mutate(A_len = str_length(Answer_pred),
         Model = "DistilBERT",
         Tok_time = tok_times$DistilBERT,
         Load_time = mod_times$DistilBERT)

data <- rbind(l_bert, d_bert)
```

Get a dataframe with both answers present:

```
df_present <- df %>%
  filter(!is.na(L_BERT_answer) & !is.na(DistilBERT_answer))
#write_csv(df_present, "tweetQA_bothpresent.csv")

d_present <- data %>%
  filter(!is.na(data$Answer_pred))
```

Evaluate data loss and processing time

Processing time summary (full dataset):

```
# Summarise
time_summary <- data %>%
  mutate(Tok_time = as.numeric(Tok_time),
         Load_time = as.numeric(Load_time)) %>%
  group_by(Model) %>%
  summarise(Missing = sum(is.na(Answer_pred)),
            Answered = sum(!is.na(Answer_pred)),
            Mean_time = mean(Time),
            'Max time' = max(Time),
            'Min time' = min(Time),
            'Total time' = sum(Time),
            'Tokenizer loading time' = max(Tok_time),
            'Model loading time' = max(Load_time),
            'Total time with loading' = sum(Time) + max(Tok_time)+ max(Load_time)) %>%
  mutate_if(is.numeric, round, 3)

time_summary
```

```
## # A tibble: 2 x 10
##   Model Missing Answered Mean_time `Max time` `Min time` `Total time`
##   <chr>   <dbl>    <dbl>     <dbl>      <dbl>      <dbl>        <dbl>
## 1 BERT      303    11475     0.429     115.        0.182        5047.
## 2 Dist…     446    11332     0.08        6.06      0.027         943.
## # … with 3 more variables: `Tokenizer loading time` <dbl>, `Model loading
## #   time` <dbl>, `Total time with loading` <dbl>
```

Processing time summary (all present dataset):

```
# Summarise
time_summary <- d_present %>%
  filter(Time<10) %>%
  mutate(Tok_time = as.numeric(Tok_time),
         Load_time = as.numeric(Load_time)) %>%
  group_by(Model) %>%
  summarise(Missing = sum(is.na(Answer_pred)),
            Answered = sum(!is.na(Answer_pred)),
            Mean_time = mean(Time),
            "sd time" = sd(Time),
            'Max time' = max(Time),
            'Min time' = min(Time),
            'Total time' = sum(Time),
            'Tokenizer loading time' = max(Tok_time),
            'Model loading time' = max(Load_time),
            'Total time with loading' = sum(Time) + max(Tok_time)+ max(Load_time)) %>%
  mutate_if(is.numeric, round, 3)

time_summary
```

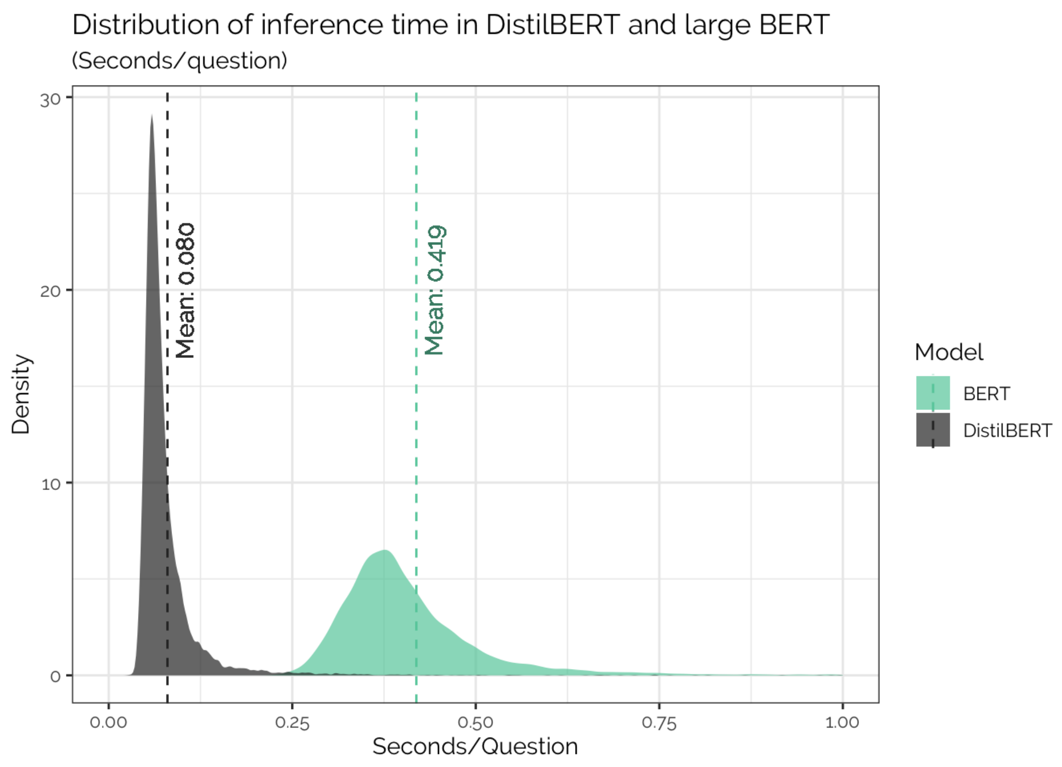
```
## # A tibble: 2 x 11
##   Model Missing Answered Mean_time `sd time` `Max time` `Min time`
##   <chr>   <dbl>    <dbl>     <dbl>     <dbl>      <dbl>      <dbl>
## 1 BERT        0    11474     0.419     0.169       4.13      0.182
## 2 Dist…       0    11332     0.08      0.101       6.06      0.027
## # … with 4 more variables: `Total time` <dbl>, `Tokenizer loading
## #   time` <dbl>, `Model loading time` <dbl>, `Total time with
## #   loading` <dbl>
```

Visualisations

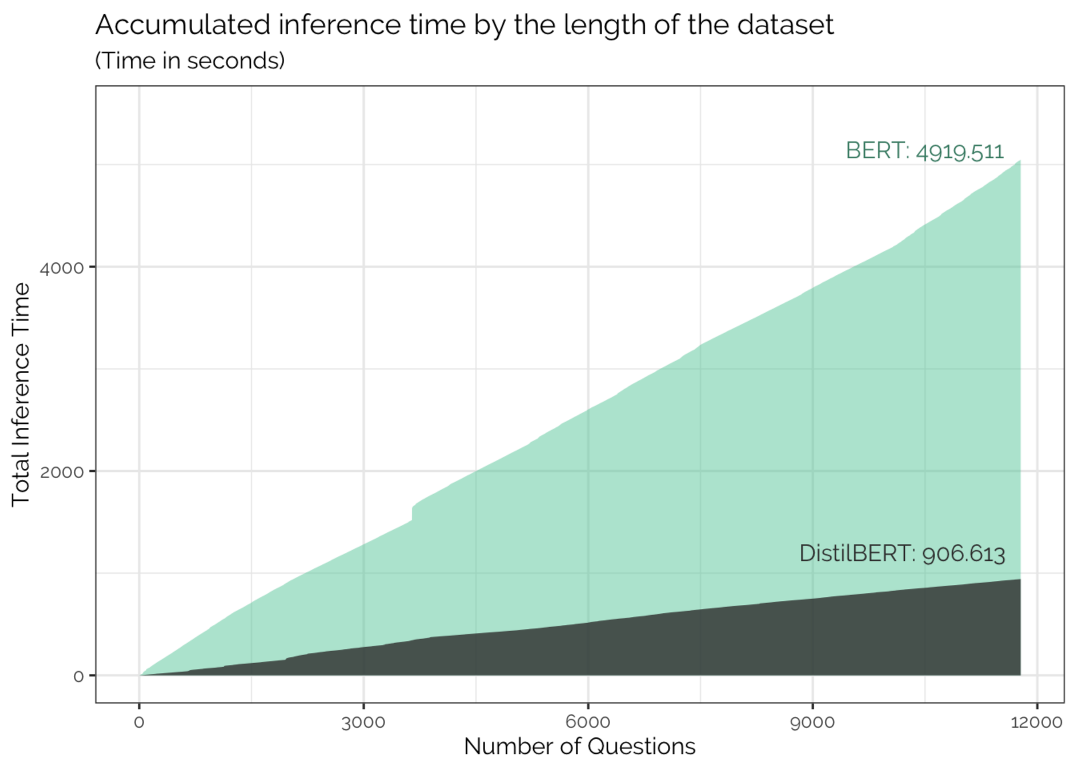
Time by model

```
d_present <- d_present %>%
  filter(Time<10)
# Density plot
density <- ggplot(d_present, aes(Time, fill = Model))+
  geom_vline(data=time_summary, aes(xintercept=Mean_time, color=Model),
             linetype="dashed")+
  geom_text(aes(x=0.080, label="\nMean: 0.080", y=20), colour="grey23", angle=90, size=4, family = "Raleway"
) +
  geom_text(aes(x=0.419, label="\nMean: 0.419", y=20), colour="aquamarine4", angle=90, size=4, family = "Ral
eway")+
  geom_density(col = NA, alpha = 0.7)+
  theme_bw()+
  scale_colour_manual(values=cp)+
  scale_fill_manual(values=cp)+
  labs( x = "Seconds/Question",
        y = "Density",
       title = "Distribution of inference time in DistilBERT and large BERT",
       subtitle = "(Seconds/question)")+
  theme(text = element_text(family ="Raleway"))+
  xlim(0,1)

density
```

```
## Warning: Removed 117 rows containing non-finite values (stat_density).
```

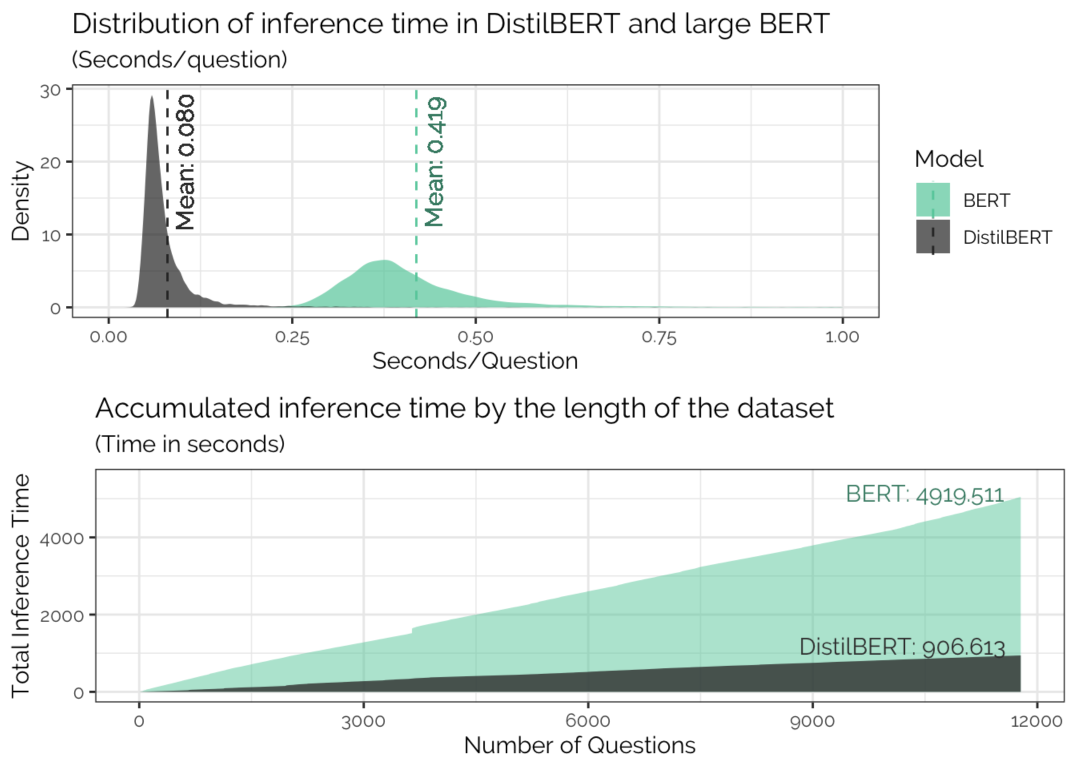


```
#Cumulative processing time plot
cumsum_l <- cumsum(l_bert$Time)
cumsum_d <- cumsum(d_bert$Time)

cumulative <-  ggplot()+
  geom_area(aes(1:length(cumsum_l), cumsum_l), fill = "aquamarine3", alpha = 0.5)+
  geom_text(aes(x=10500, label="BERT: 4919.511", y=5150), colour="aquamarine4", family = "Raleway", hjust="c
enter") +
  geom_text(aes(x=10200, label="DistilBERT: 906.613", y=1200), colour="grey23", family = "Raleway", hjust="c
enter") +
  geom_area(aes(1:length(cumsum_d), cumsum_d), fill = "grey23", alpha = 0.8)+
  theme_bw()+
  labs(x = "Number of Questions",
       y = "Total Inference Time",
       title = "Accumulated inference time by the length of the dataset",
       subtitle = "(Time in seconds)")+
  theme(text = element_text(family = "Raleway"), legend.title=element_blank())+
  ylim(0,5500)

cumulative
```



```
time_plots <- gridExtra::grid.arrange(density,cumulative, nrow=2)
```

```
## Warning: Removed 117 rows containing non-finite values (stat_density).
```



```
ggsave("timeplots.png", time_plots, width = 8, height = 6)
```

Linear models for stats

```
# Predict time by model
time_by_model <- lm(Time ~ Model, d_present)
summary(time_by_model)
```

```
##
## Call:
## lm(formula = Time ~ Model, data = d_present)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.2365 -0.0354 -0.0177  0.0079  5.9836
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.418769   0.001302   321.6   <2e-16 ***
## ModelDistilBERT -0.338765   0.001847  -183.4   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1395 on 22804 degrees of freedom
## Multiple R-squared:  0.5959, Adjusted R-squared:  0.5958
## F-statistic: 3.362e+04 on 1 and 22804 DF,  p-value: < 2.2e-16
```

```
t.test(Time~Model, d_present)
```

```
##
##  Welch Two Sample t-test
##
## data:  Time by Model
## t = 183.91, df = 18776, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.3351542 0.3423754
## sample estimates:
##       mean in group BERT mean in group DistilBERT
##               0.41876947               0.08000467
```

Manual evaliation of performance metrics GLEU and METEOR

```
d100 <- read_csv2("/Users/anitakurm/Downloads/df_samples_scores100_judged.csv")
```

```
## Using ',' as decimal and '.' as grouping mark. Use read_delim() for more control.
```

```
## Warning: Missing column names filled in: 'X1' [1], 'X11' [11]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   BERT = col_character(),
##   DistilBERT = col_character(),
##   Gold_1 = col_character(),
##   Gold_2 = col_character(),
##   BERT_METEOR = col_double(),
##   DistilBERT_METEOR = col_double(),
##   BERT_GLEU = col_double(),
##   DistilBERT_GLEU = col_double(),
##   JUDGE = col_double(),
##   X11 = col_character()
## )
```

```
d100 <- d100 %>%
  mutate(Gold1_length = sapply(strsplit(Gold_1, " "), length),
         Gold2_length = ifelse(is.na(Gold_2), 0, sapply(strsplit(Gold_2, " "), length)),
         Gold_max_length = ifelse(Gold1_length>Gold2_length, Gold1_length, Gold2_length),
         Meteor_correct = ifelse(JUDGE == 0, 1, 0),
         length_cat = ifelse(Gold_max_length <= 2, " up to 2 (incl)", "greater than 2"),
         length_cat2 = ifelse(Gold_max_length <= 3, " up to 3 (incl)", "greater than 3"))

d100 %>%
  group_by(X11) %>%
  summarise(n())
```

```
## # A tibble: 3 x 2
##   X11   `n()`
##   <chr> <int>
## 1 C         6
## 2 F        11
## 3 <NA>     83
```

```
d100_compare <- d100 %>%
  filter(!is.na(JUDGE))

d100_compare %>%
  summarise('GLEU correct' = sum(JUDGE),
            'METEOR correct' = sum(Meteor_correct))
```

```
## # A tibble: 1 x 2
##   `GLEU correct` `METEOR correct`
##            <dbl>            <dbl>
## 1             48               34
```

```
d100_compare %>%
  group_by('Gold Standard Length' = length_cat) %>%
  summarise('GLEU correct' = sum(JUDGE),
            'METEOR correct' = sum(Meteor_correct))
```

```
## # A tibble: 2 x 3
##   `Gold Standard Length` `GLEU correct` `METEOR correct`
##   <chr>                           <dbl>            <dbl>
## 1 " up to 2 (incl)"                  39               21
## 2 greater than 2                      9               13
```

```
d100_compare %>%
  group_by('Gold Standard Length' = length_cat2) %>%
  summarise('GLEU correct' = sum(JUDGE),
            'METEOR correct' = sum(Meteor_correct))
```

```
## # A tibble: 2 x 3
##   `Gold Standard Length` `GLEU correct` `METEOR correct`
##   <chr>                           <dbl>            <dbl>
## 1 " up to 3 (incl)"                  42               26
## 2 greater than 3                      6                8
```

```
d100_compare %>%
  group_by('Gold Standard Length' = Gold_max_length) %>%
  summarise('GLEU correct' = sum(JUDGE),
            'METEOR correct' = sum(Meteor_correct))
```

```
## # A tibble: 7 x 3
##   `Gold Standard Length` `GLEU correct` `METEOR correct`
##                    <dbl>          <dbl>            <dbl>
## 1                      1             19                6
## 2                      2             20               15
## 3                      3              3                5
## 4                      4              4                2
## 5                      5              1                2
## 6                      6              0                3
## 7                      7              1                1
```

Applying METEOR for long answers dataset and GLEU for short answers dataset

```
# Read in the data
long = read.csv("df_long_answers.csv")
short = read.csv("df_short_answers.csv")
# Take only necessary columns
long = select(long,7:10)
short = select(short,7:10)

# re-define color palette so it's consistent across plots
cp <- c("aquamarine3", "grey19")
```

Long

```
bert = select(long,1,3)
bert['Model'] = 'BERT'
names(bert)[1] <- "METEOR"
names(bert)[2] <- "GLEU"
distil = select(long, 2,4)
distil['Model'] = 'DistilBERT'
names(distil)[1] <- "METEOR"
names(distil)[2] <- "GLEU"
data = rbind(bert, distil)
mu <- plyr::ddply(data, "Model", summarise, grp.mean=mean(METEOR))
mu
```

```
##        Model  grp.mean
## 1       BERT 0.4334346
## 2 DistilBERT 0.3874651
```

```
meteor <- ggplot(data, aes(x=METEOR, fill=Model)) +
  geom_density(col = NA, alpha=0.6, position="identity") +
  geom_vline(data=mu, aes(xintercept=grp.mean, color=Model),
             linetype="dashed") +
  geom_text(aes(x=0.387, label="\nMean: 0.387", y=2.5), colour="grey23", angle=90, size=4, family = "Raleway
") +
  geom_text(aes(x=0.433 , label="\nMean: 0.433  ", y=2.5), colour="aquamarine4", angle=90, size=4, family =
"Raleway")+
  theme_bw() +
  ylim(0,3.5)+
  theme(text = element_text(family = "Raleway"))+
  scale_colour_manual(values=cp)+
  scale_fill_manual(values=cp)+
  labs(x = "METEOR score",
       y = "Density")

meteor_by_model <- lm(METEOR ~ Model, data)
summary(meteor_by_model)
```

```
##
## Call:
## lm(formula = METEOR ~ Model, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.43343 -0.32549 -0.01586  0.27816  0.61253
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.433435   0.005289  81.945  < 2e-16 ***
## ModelDistilBERT -0.045970   0.007480  -6.145 8.37e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3269 on 7638 degrees of freedom
## Multiple R-squared:  0.00492,    Adjusted R-squared:  0.00479
## F-statistic: 37.77 on 1 and 7638 DF,  p-value: 8.374e-10
```

```
t.test(METEOR ~ Model, data)
```

```
##
##  Welch Two Sample t-test
##
## data:  METEOR by Model
## t = 6.1454, df = 7637.8, p-value = 8.374e-10
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.03130618 0.06063291
## sample estimates:
##      mean in group BERT mean in group DistilBERT
##               0.4334346                0.3874651
```

Short

```
bert = select(short,1,3)
bert['Model'] = 'BERT'
names(bert)[1] <- "METEOR"
names(bert)[2] <- "GLEU"
distil = select(short, 2,4)
distil['Model'] = 'DistilBERT'
names(distil)[1] <- "METEOR"
names(distil)[2] <- "GLEU"
data = rbind(bert, distil)
mu <- plyr::ddply(data, "Model", summarise, grp.mean=mean(GLEU))
mu
```

```
##        Model  grp.mean
## 1       BERT 0.6143202
## 2 DistilBERT 0.5349806
```

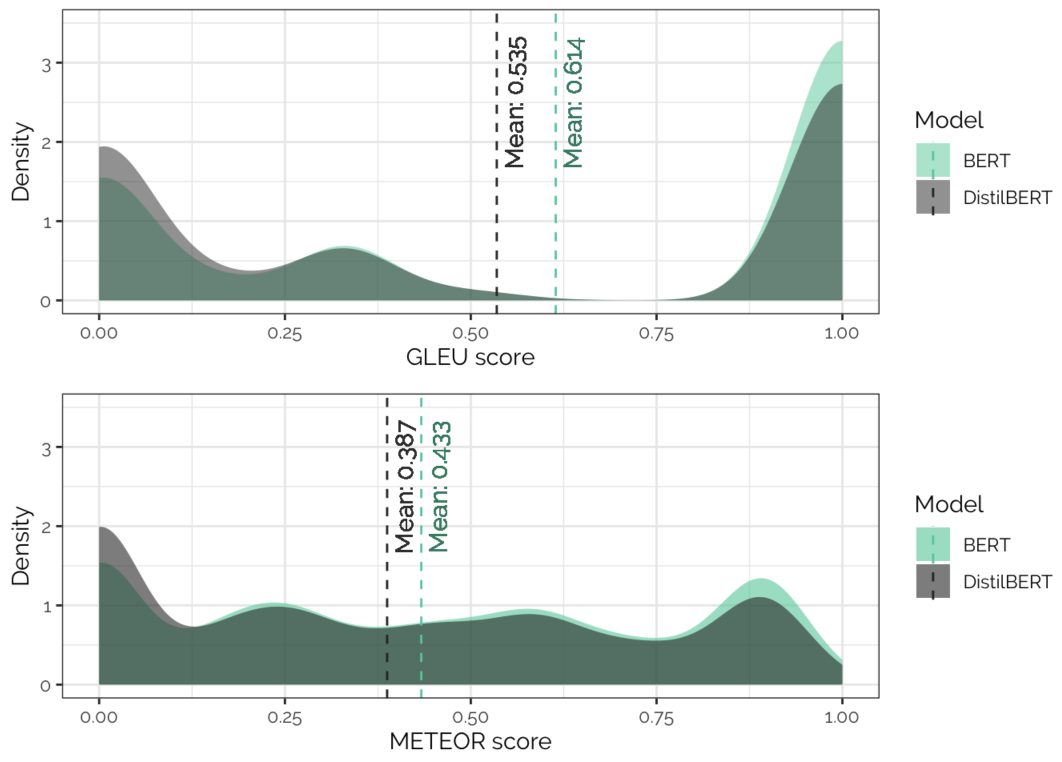
```
gleu <- ggplot(data, aes(x=GLEU, fill=Model)) +
  geom_density(col = NA, alpha=0.5, position="identity") +
  geom_vline(data=mu, aes(xintercept=grp.mean, color=Model),
             linetype="dashed") +
  geom_text(aes(x=0.535, label="\nMean: 0.535", y=2.5), colour="grey23", angle=90, size=4, family = "Raleway
") +
  geom_text(aes(x=0.614 , label="\nMean: 0.614  ", y=2.5), colour="aquamarine4", angle=90, size=4, family =
"Raleway")+
  theme_bw() +
  xlim(0,1) +
  ylim(0,3.5)+
  theme(text = element_text(family = "Raleway"))+
  scale_colour_manual(values=cp)+
  scale_fill_manual(values=cp)+
  labs(x = "GLEU score",
       y = "Density")

gleu_by_model <- lm(GLEU ~ Model, data)
summary(gleu_by_model)
```

```
##
## Call:
## lm(formula = GLEU ~ Model, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.6143 -0.5350  0.3857  0.3857  0.4650
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.614320   0.005330  115.27   <2e-16 ***
## ModelDistilBERT -0.079340   0.007537  -10.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4536 on 14488 degrees of freedom
## Multiple R-squared:  0.00759,    Adjusted R-squared:  0.007522
## F-statistic: 110.8 on 1 and 14488 DF,  p-value: < 2.2e-16
```

```
t.test(GLEU ~ Model, data)
```

```
##
##  Welch Two Sample t-test
##
## data:  GLEU by Model
## t = 10.526, df = 14477, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.06456574 0.09411336
## sample estimates:
##       mean in group BERT mean in group DistilBERT
##                0.6143202                0.5349806
```

```
met_gl <- gridExtra::grid.arrange(gleu, meteor, nrow = 2)
```



```
ggsave("meteor_gleu.png", met_gl,width = 9, height = 8)
```

Gather citations

```
citation()
```

```
##
## To cite R in publications use:
##
##   R Core Team (2019). R: A language and environment for
##   statistical computing. R Foundation for Statistical Computing,
##   Vienna, Austria. URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2019},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please
## cite it when using it for data analysis. See also
## 'citation("pkgname")' for citing R packages.
```

```
citation("tidyverse")
```

```
##
## To cite package 'tidyverse' in publications use:
##
##   Hadley Wickham (2017). tidyverse: Easily Install and Load the
##   'Tidyverse'. R package version 1.2.1.
##   https://CRAN.R-project.org/package=tidyverse
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {tidyverse: Easily Install and Load the 'Tidyverse'},
##     author = {Hadley Wickham},
##     year = {2017},
##     note = {R package version 1.2.1},
##     url = {https://CRAN.R-project.org/package=tidyverse},
##   }
```

```
citation("rjson")
```

```
##
## To cite package 'rjson' in publications use:
##
##   Alex Couture-Beil (2018). rjson: JSON for R. R package version
##   0.2.20. https://CRAN.R-project.org/package=rjson
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {rjson: JSON for R},
##     author = {Alex Couture-Beil},
##     year = {2018},
##     note = {R package version 0.2.20},
##     url = {https://CRAN.R-project.org/package=rjson},
##   }
##
## ATTENTION: This citation information has been auto-generated from
## the package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
```

```
citation("extrafont")
```

```
##
## To cite package 'extrafont' in publications use:
##
##   Winston Chang, (2014). extrafont: Tools for using fonts. R
##   package version 0.17.
##   https://CRAN.R-project.org/package=extrafont
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {extrafont: Tools for using fonts},
##     author = {Winston Chang,},
##     year = {2014},
##     note = {R package version 0.17},
##     url = {https://CRAN.R-project.org/package=extrafont},
##   }
##
## ATTENTION: This citation information has been auto-generated from
## the package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
```