UCLouvain

epl

École polytechnique de Louvain

Extracting ESG data from business documents

Author: **Jérôme Van Der Elst**
Supervisor: **Siegfried Nijssen**
Readers: **Nicolas Golenvaux, Victor Hamer**
Academic year 2020–2021
Master [120] in Computer Science and Engineering

Abstract

Documents are a natural way for humans to share information. They work very well to represent data in a variety of formats such as text, tables, graphs, titles, footnotes etc. They are, however, time-consuming to read through and not suited as input for computer driven analysis. It is natural to ask ourselves whether this unstructured data can be extracted into a structured format to drive algorithms that perform quantitative analysis. This is a challenging task as heuristic based methods are often tricky to implement and don't generalize well, whereas natural language processing techniques don't have the tools to capture the layout information necessary to correctly interpret these documents.

This work studies different techniques for data extraction from business documents in the particular case of extracting CO2 emissions and compares their weaknesses in order to propose improvements.

Acknowledgements

I would like to sincerely thank my thesis advisor Professor Siegfried Nijssen who helped me weekly during my work and was always helpful when I ran in trouble or had a question about my writing. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it. I would like to thank him for accepting the subject, as it allowed me to read many papers in a very interesting field. Finally, I would also like to thank my parents for their general feedback on this master thesis.

Contents

Chapter 1

Introduction

Every year, companies release an enormous amount of interesting data through their business documents. Business documents are files that provide details related to a company's internal and external proceedings, and may be digital-born or in scanned format. Examples of such business documents are financial reports, lawsuit documents, letters to the investors, ... They are written in natural language and can be organized in a variety of ways from text, table or figures. Transforming them into structured data is challenging due to variance in format, cross references and complex document structures.

Many companies extract data from business documents manually. Efforts of automation have resulted in semi-automated approaches where specific rules for specific categories of documents are used by software to extract the data. These rules need to be hard-coded by humans and adapted to different documents. Recently, research has been conducted to apply deep learning to completely automate this task, in particular for invoice data extraction. Companies that engage in data extraction from documents can be separated in two categories. The first type are companies that want to digitalize their data. They receive many invoices, receipts or forms in paper format which need to be encoded in their database. The second type are data providers. Data providers are companies whose business model is to sell data. They collect their data by either extracting it manually or using internally developed tools.

As the bid to invest responsibly has become mainstream over the past few years, the demand for environmental, social and governance (ESG) data has skyrocketed. An example of ESG data would be the total annual CO2 emissions of a company or the gap between the highest salary and the lowest salary. Examples of data providers in this sector are MSCI, Data Simply, ISS, Owl Analytics, SenseFolio and TrueValueLabs. Some of these companies, such as TrueValueLabs, have started applying Artificial Intelligence to parts of the data extraction pipeline. However, most companies still rely on extensive manual work to extract this information, which explains the extremely high subscription fees.

There is no doubt that making ESG data widely available and allowing for easy comparison between companies would be immensely beneficial for both the economy and the planet.

Starting from this very ambitious context, this thesis will investigate further on the more specific question of "Can we extract CO2-related data automatically from PDF sustainability documents released by companies". This will serve the purpose of first demonstrating the difficulties encountered when working with PDF documents and then to investigate the current technologies and how they can be adapted to this task.

There currently does not exist any ESG or CO2 dataset that could be used for training a machine learning model. So this thesis will explore the possibilities under the constraint of a limited amount of available training data.

After the ESG data has been extracted, it can be used for comparative analysis. For example, one could normalize the CO2 data by industry to allow for a fair comparison between companies. However, this scoring mechanism is bound to have some kind of subjectivity and is therefore not the aim of this thesis. The focus will be set on the extraction of the data from PDF-based documents.

Text extraction is usually done using Natural Language processing techniques which have seen massive improvements in the last few years. Language models have reached super-human accuracy on text-based question answering datasets such as the Stanford Question Answering Dataset (SQuAD). However for extracting data from layout intensive documents, there is still no match for human performance. That is why CO2 is chosen as extraction task. CO2 data is often reported in tables, which make traditional language models unsuited for the a task. This thesis will then survey the options for automatically extracting this data, look at whether one can adapt language based deep learning methods in order for them to understand tabular data and finally look at the research on this new field called Document Intelligence, which refers to a research topic for automatically understanding and analyzing business documents (*Document Intelligence Workshop at NeuriPS 2019* 2021).

1.1 Questions

The experiments conducted in this work will try to answer the following questions:

- What is the best method for extracting CO2 data, without large training set, from PDF business documents?
- How could the selected method be improved?
- What method looks the most promising in the close future?
- Can traditional language models be adapted to tabular data, and in doing so, can we leverage all the existing open source libraries ?

1.2 Content overview

The second chapter describes current state of research regarding text extraction, language modeling, information retrieval systems and some concepts of computer vision.

The third chapter details how a small CO2-dataset was built for testing purposes. It then explains the three implementations that were compared against in the experiments. The first algorithm interprets the PDF in a grid-format and is inspired from table extraction techniques. The second method preprocesses the PDF documents into a textual representation that tries to maintain as much layout information as possible and is then introduced to a question answering system. This approach exploits the advantage abundant open source software on question answering and text extraction systems. The last implementation tests a new layout aware language model called LayoutLM designed for Document Intelligence tasks on the CO2 dataset.

The fourth chapter contains the experiments with the three implementations.

And finally, chapter 6 contains a conclusion and an answer to the questions that were introduced in this chapter.

Chapter 2

Related work

Most document information retrieval is still done manually. Semi-automated tools like DocParsesr(*Docparser - Document Parser Software* 2021) can be used that allow to define custom document-specific rules and templates which then drive a rule-base extraction system.

Another option is to use **table extraction** software, which is also semi-automated as it usually requires postprocessing steps to clean and format the table.

Natural language processing techniques are used when the the data that has to be extracted is mainly represented in textual format, but these techniques are unsuited when the data is in the form of tables or figures.

(Xu et al. 2019) define Document AI, or Document Intelligence as a relatively new research topic that refers techniques for automatically reading, understanding, and analyzing business documents. The task that is often studied in this context is invoice extraction, which consists for example in retrieving the TVA number from a scanned invoice. **Document layout analysis** (DLA), the goal of which is to identify blocks of text and non-textual objects and classify them accordingly, is a standard approach in document intelligence. For each class, dedicated extraction software can then be used to further process the data. Notable systems that use this pipeline are Google's document AI tools (*Document AI Solution Google Cloud* 2021), Amazon's text extraction platform (*Documents and Block Objects - Amazon Textract* 2021), Microsoft's form recognizer (*Form Recognizer Microsoft Azure* 2021) and IBM's document question answering system (*Watson Discovery IBM* 2021). More recently, researchers have tried to understand document in their whole by creating models that understand both text and layout. This involves adapting language models to handle layout-related input features and is referred to as **layout-aware language modeling**.

2.1 Collecting data

The first step in any information extraction system is to acquire the data from which the information can be extracted. Some websites contain large publicly accessible databases of documents, but most documents are scattered around the world wide web. The collection of these documents can either be done manually or automatically. Automating this process using software that crawls the web is referred to as web scraping.

Different approaches are used for webscraping. A first type of webscrapers are HTML-parsers which parse the HTML code returned by the server. Examples of such libraries are BeatifulSoup and Scrapy. A second breed of scrapers have a full-fledged web browser

embedded and can therefore directly manipulate the DOM interface. These scrapers are more powerful and more suited for heavy dynamic pages as they can execute Javascript code and completely load the webpage. Example of such scrapers are Selenium and Puppeteer. Scrapers have the option of either starting from a web-page and then recursively click to access new content or use a search engine to query relevant websites.

Web scraping is a relatively new concept and is still in the gray area of legality. BERT (a popular language model) for example was trained on the entire Wikipedia (2,500 million words) and Book Corpus (800 million words). However, both these datasets are open source and allow for this type of use. What about business documents for which the authoring company might not necessarily agree with the use of their documents ?

(Krotov and Silva 2018) introduce the following questions in order to determine whether webscraping is ethical and therefore likely to be legal:

- Is Web Crawling or Web Scraping explicitly prohibited by the website's "terms of use" policy?
- Is the website's data explicitly copyrighted?
- Does the project involve illegal or fraudulent use of the data?
- Can crawling and scraping potentially cause material damage to the website or Web server hosting the website ?
- Can the data obtained from the website compromise individual privacy?
- Can the data obtained from the website reveal confidential information about operations of the organizations providing data or the company owning the website?
- Can the project requiring the Web data potentially diminish the value of the service provided by the website?

As the answer to all of those questions is "No" for our work, it passes the checklist and webscraping can be considered for the project.

Web scraping is one of the main contributing factors to the fast development of data intensive machine learning applications. Common Crawl (*Want to use our data? – Common Crawl* 2021) for example is an open repository of web crawl data consisting of web page data, metadata extracts and text extracts. It is obtained by scraping web pages and ignoring HTML markup. It produces around 20TB of scraped data each month. The organization started crawling the web in 2008 and the data has reached as size of 320 TB in April 2021. Even though such data often contains gibberish text like menus or error messages, the amount of data that can be obtained in such a way is unprecedented.

2.2 Text mining

Text mining is the discovery by computers of new, previously unknown information, by automatically extracting information from different written resources (*What Is Text Mining, Marti Hearst* 2005). Text mining can be applied on textual input data in order to create a structured database on which data mining techniques can be applied. Text mining techniques include from counting frequent words, building graphs of word dependencies to study certain properties of a text, logistic regressions, decision trees, k-means to the

recent large and complex language models. Applications include document classification, document clustering, sentiment analysis, document summarizing, document retrieval, information retrieval, ... Finding an appropriate method for extracting CO2 data can be considered a text mining task which is why this field is likely to contain the right tools for solving this problem.

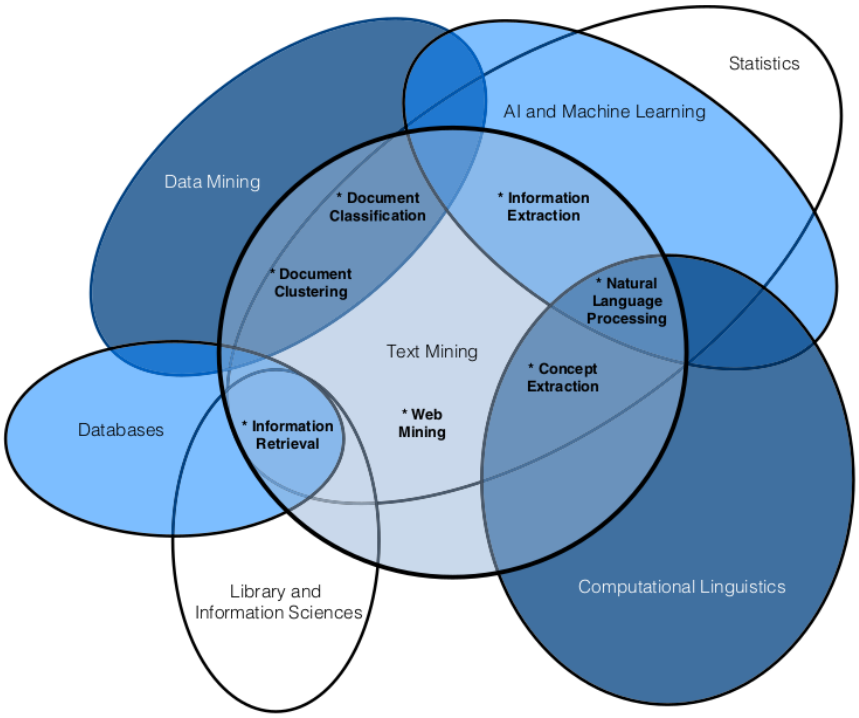


Figure 2.1: Venn dagram form"Practical Text Mining and Statistical Analysis for Non-Structured Text Data Applications, G. Miner, 2012 gives an overview of the different fields that overlap with text mining. The particular fields that of interest to this work are Information Retrieval, Information Extraction and Natural Language Processing

2.2.1 Language Modeling

Text mining uses Natural Language Processing to turn free unstructured text from documents into normalized, structured data suited for analysis or to drive machine learning algorithms.

The main component of natural language processing (NLP) is the language model. The goal of language modeling is to estimate the probability distribution of various linguistic units such as words, sentences, paragraphs, etc. Language modeling techniques can be separated in two categories: count-based and continuous space (Synced 2017). The count-based methods, such as traditional statistical models, usually involve making an n-th order Markov assumption and estimating n-gram probabilities via counting and subsequent smoothing. In recent years, continuous-space language models such as feed-forward neural probabilistic language models and recurrent neural network language models are proposed. These neural models solve the problem of data sparsity of the n-gram model, by representing words as vectors and using them as input for downstream tasks. The parameters are learned as part of the training process. Word embeddings exhibit the property whereby semantically close words are likewise close in the induced vector

space. Moreover, neural language models can also capture the contextual information at the sentence-level, corpus-level and sub-word level.

Tokenizers

The granularity at which models interpret language depends on the type of tokenization that is used. Tokenization is responsible for cutting text into smaller pieces used as input to a language model. Depending on the task and the language, different tokenization mechanisms are used. This can be either words (word tokenization), subwords (subword tokenization) or characters (character tokenization)

In our scenario for example, we would probably want to partition a PDF document into sub-tokens such that "emission" and "emissions" would share some representation. Sentences should be split on white spaces, dots and commas. But what about numbers ? We don't want 956,587 to be split into 986 and 587 as it does not make sense to interpret numbers at the subword level. It is therefore interesting to study how tokenization is commonly performed in language models.

Word tokenization and character tokenization are straightforward, but have their weaknesses. Word tokenization suffers from "out of vocabulary (OOV) words", i.e. words that have never been seen during training need to be replaced by $< UNK >$. Character tokenization handles OOV but words makes it much harder for the model to learn meaningful representations. Subword tokenization is the middle ground and what currently works best. WordPiece, BPE and Unigram Langauge Modeling are the commonly used methods in NLP tasks. Tokenization happens at the subword level and OOV words are impossible.

Algorithm 1 WordPiece (Schuster and Nakajima 2012)

1: Initialize dictionary with a,b,c,...,z
2: Build a language model on this dictionary and training corpus
3: Add a new word to the dictionary by combining two current units in the dictionary. Choose this new word unit out of all possible ones that increase the likelihood on the training data the most when added to the model
4: Repeat 3 for a predefined number of iterations

Adding a new wordpiece in a brute force way by testing all possible combinations and the effect on the corpus likelihood by the language model is expensive ($O(n^2)$). Especially if the language model itself is expensive. Therefore WordPiece implements several optimizations such as only merging wordpieces if they are present in the training corpus, only testing high potential pairs, and combining several clustering steps in one step. Also note that for this type of tokenization, a special character is usually appended or prepended to words to indicate where the initial space characters were. In the original paper, they use 3 to 5-gram language models with Katz back-off smoothing. BERT (introduced later) uses WordPiece model and use the special characters ## to indicate where a word was splitted into two distinct tokens.

Algorithm 2 BPE (Sennrich, Haddow, and Birch 2016)

1: Initialize dictionary with a,b,c,...,z
2: Count the frequencies of all possible sub-words in the training corpus
3: Add a new word to the dictionary by combining two current units in the dictionary. Choose the new word unit out of all possible that has the highest frequency
4: Repeat 3 for a predefined number of iterations

Byte Pair Encoding was initially used as a compression algorithm and is very close to WordPiece, but instead of maximizing the likelihood over some language model, it maximizes the character n-gram frequency. BPE is used by RoBERTa (introduced later).

Transformer model

One of these neural probabilistic language models is the Transformer introduced by (Vaswani et al. 2017). In recent years this model and its attention mechanism have become ubiquitous in NLP. Most current state of the art NLP technologies are based on the principles of this architecture. It is shortly introduced here as it will help explain BERT, on which the question answering system is based. The Transformer is a Seq2Seq model that follows the encoder-decoder principle and uses embedding layers, linear layers, attention layers and layer normalization. The main innovation of the Transformer is the use of a new type of layers called "attention layers" combined with a positional encoding that performs surprisingly well on language tasks.

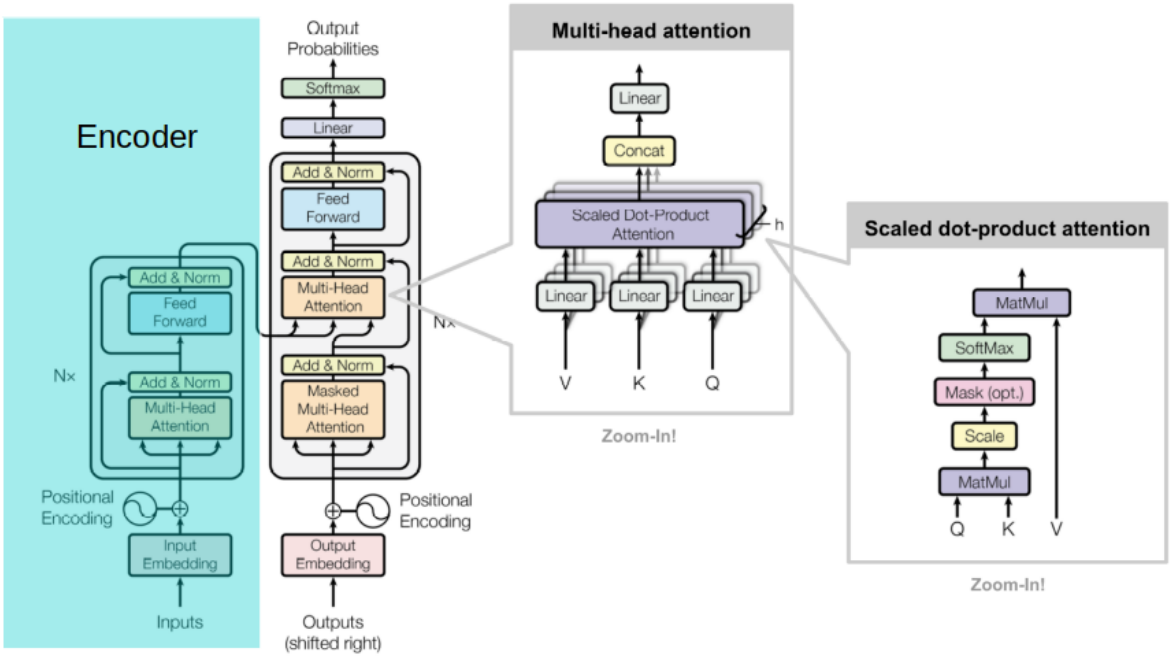


Figure 2.2: Transformer diagram showing the model composed of an encoder (left) feeding into a decoder (right) which are both based on attention layers. We are interested in the encoder for this thesis. (*Attention? Attention!* 2018)

The encoder produces embeddings in some encoding vector space, which are then decoded by the decoder in a recurrent/autoregressive way. The encoder is of interest for this work because it produces contextualized word embeddings. So in the sentences

"He robbed a bank" and "He sat on a bank of sand", this architecture will produce two different word embeddings for the word "bank", because it appears in a different context. Other word embedding methods such as Word2Vec and Glove would produce the same word embedding for "bank" in both cases.

An attention layer applies the following formula to a Query $Q \in R^{\#words \times dimension}$, a Key $K \in R^{\#words \times dimension}$ and a value $V \in R^{\#words \times dimension}$, which are all vectors coming from the same word embedding, but passed through a different linear transformation:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V}$$

where Softmax is applied row by row, and $\sqrt{n}$ is a scaling factor used to ensure that the Softmax does not go too much in saturation region. This formula shows that every new embedding is computed as a weighted average of all the previous embeddings in V, weighted by attention scores derived from the dot product matching of Q and K. This allows it to build a context dependent representation of words. In practice however, they use multi-headed attention, which is just a concatenation of different attention layers in parallel.

The inputs of the model are words, which need to be transformed to vectors before being fed into a neural network. This is done thanks to a so called embedding layer. An embedding layer can be seen as a linear layer that takes as input the one hot representation of the word, given by the dictionary that was produced during tokenization. In practice however it is a faster trainable lookup table implementation.

The attention mechanism has one weakness: it does not retain positional information. One can be convinced of this by swapping to words in the input sequence, which will result in the corresponding rows in Q K and V to be swapped, but the end attention scores will be the same, as well as the output of the layer. Therefore, the authors of the Transformer architecture add a positional encoding to the input word embeddings in the following way.

For a position t, the value of the dimension i of the positional embedding is given by:

$$\vec{PE_t}^{(i)} = \begin{cases} \sin(\omega_k.t), & \text{if } i = 2k \\ \cos(\omega_k.t), & \text{if } i = 2k+1 \end{cases} \quad \text{with } \omega_k = \frac{1}{10000^{2k/d}}$$

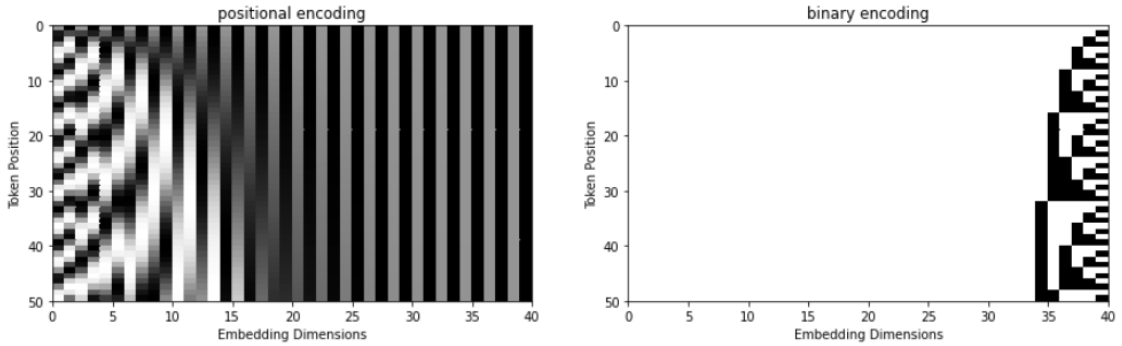


Figure 2.3: The positional encoding used by the Ttransformer (left) can be seen as the continuous counterpart of binary positional encoding (right). This image explains why positional embeddings can just be added to the word embeddings: only the first dimensions to the left are used, the rest of the dimensions can be used for the word embeddings.

The final embedding that the Transformer takes as input is a simple point-wise addition of the position embedding and the (learned) word embedding.

$$Embedding = PE(word) + Embedding(word)$$

This positional embedding implies a 1D interpretation of text. In layout-aware language models, the 1D positional embedding is replaced with 2D positional embeddings, which in practice is often the sum of two 1D embeddings from the X and Y dimension. We experiment with such a model called LayoutLM, which will be introduced later.

BERT

BERT stands for Bidirectional Encoder Representations from Transformers (Devlin et al. 2019) and is used for many natural language processing tasks . The BERT architecture is inspired from the transformer model. It is equivalent to the encoder of the transformer but with different hyperparameters and the addition of segment embeddings to the word embeddings. The output embedding of the encoder are used as input to a classification layer for performing various downstream NLP tasks.

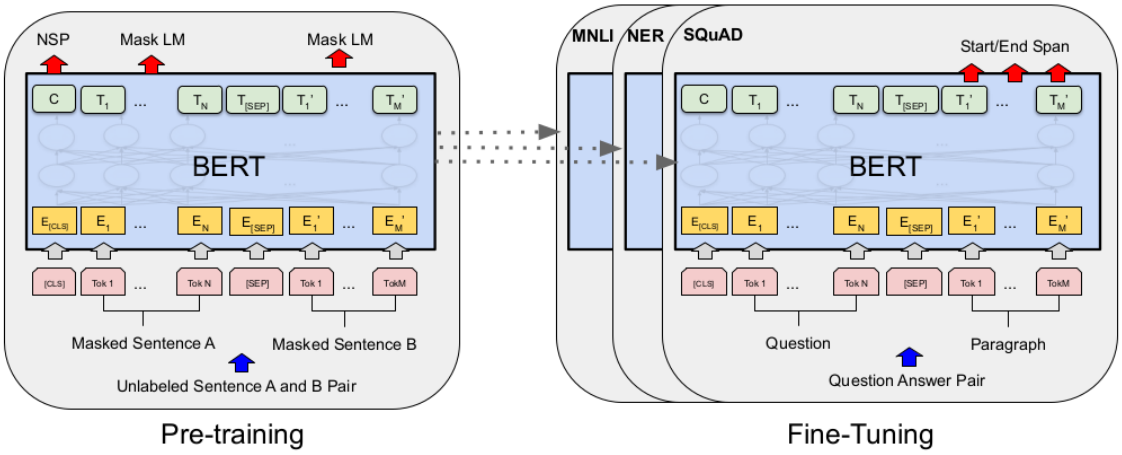


Figure 2.4: Bert architecture based on the transformer encoder. E... indicates an embedding of the corresponding token. T... indicates a contextualized BERT embedding of the corresponding token. [CLS] token stands for classification is special token for sentence-level representation. [SEP] are used for separation. BERT can be finetuned for question answering, named entity recognition, document classification, ...

BERT takes as input a concatenation of two segments $x_1, ..., x_N$ and $y_1, ..., y_M$. The two segments are presented as a single input sequence to BERT with special tokens delimiting them.

$$[CLS], x_1, ..., x_N, [SEP], y_1, ..., y_M, [SEP]$$

The model is first pretrained in an unsupervised way on a large unlabeled text corpus and subsequently finetuned using end-task labeled data. In conclusion, BERT is a way to create bidirectional contextualized word embeddings that make it possible to fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks.

(Liu et al. 2019) introduce a new way of retraining BERT called RoBERTa. They found that BERT was significantly undertrained and could outperform many of the models

that were published after it by training it longer, removing the next sentence prediction objective, training on longer sequences and using dynamic masking. They also replace the WordPiece tokenization by Byte Pair Encoding (BPE). The question answering system used in this work uses RoBERTa that is finetuned on a question answering dataset called SQuAD2.

Layout Aware Language Modeling: BERTGrid, LayoutLM, LAMBERT, TILT

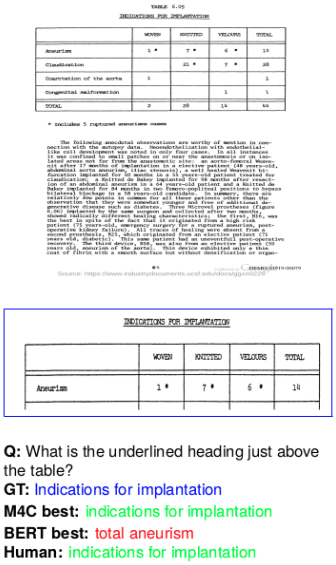
Transformer-based language models need a positional encoding added to the input embeddings in order to be able to process sequential information. In the case of the Transformer, they use a sin/cos positional encoding. Layout aware language models are a new breed of language models where researchers have modified the positional encoding to a more general 2D encoding. Instead of treating text as a 1 dimensional input stream, one could treat text as a two dimensional field with every word having the coordinate in the document from which the text was extracted. Instead of having a single positional embedding, one would have two positional embeddings for the dimension X and Y. Language models can also be augmented with visual features from the input document.

(Denk and Reisswig 2019) propose BertGrid, where they create a 2D-grid of embeddings coming from a BERT model and pass this input through a convolutional encoder-decoder model with semantic segmentation and bounding box regression. (Xu et al. 2019) propose LayoutLM, an augmented version of BERT with 2D positional embeddings as well as feature maps of the corresponding image produced by a Faster R-CNN. (Garncarek et al. 2021) propose LAMBERT, an augmented version of the RoBERTA model with 2d positional embeddings and a relative attention bias. (Powalski et al. 2021) propose TILT, an augmented version of the google T5 architecture with 2D positional embeddings.

LayoutLM will be compared against other methods on the task of extracting CO2 data from PDF documents.

DocVQA

DocVQA is a dataset for visual question answering that consists of 50,000 questions defined on 12,000+ document images. Human performance is 94.36% for exact match accuracy and the current best model as of June 2021 is TILT with an exact match score of 87.05%. DocVQA contains many types of documents such as letters, forms, reports, agenda's, tables, scientific publications, graphics, emails, notes, budgets, invoices, ... In particular it contains around 400 tables.



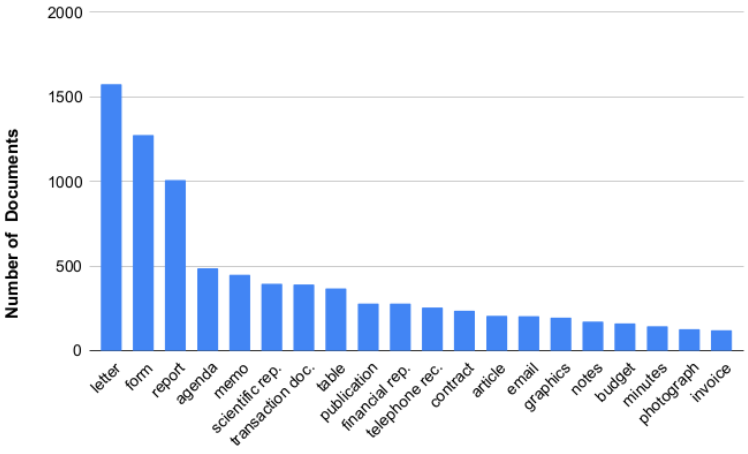


Figure 2.5: DocVQA is a heterogenous dataset of business documents containing many different types of documents.

2.2.2 Information Retrieval

IR is the activity of obtaining some information resources relevant to an information need from within large collections and is a field used within text mining. Neural IR refers to the application of shallow or deep neural networks to these retrieval tasks.

Document retrieval

Question answering systems based on neural networks are computationally very expensive. Therefore, when querying information from a large database of text, an index-based method is often used to prune out the majority of documents and present to the neural network only the most promising candidates. In such a context the neural network is called the reader and the pruning technique is called the retriever.

Some commonly used retrievers are:

1. TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying the term frequency in the document with the inverse document frequency of the word across a set of documents. It works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word. This is done to eliminate common words such as "the", "a", "and", "or", ... So a word is only considered relevant if it appears a lot of times in the document but not in other documents.

$$TF - IDF(t, d, D) = TF(t, d) * IDF(t, D)$$
$$TF(t, d) = log(1 + freq(t, d))$$
$$IDF(t, D) = log(\frac{N}{count(d \in D : t \in d)})$$

where t is a word, d a document and D the document collection. For multiple word queries, averaging is applied. Logarithms are used to dampen the effect of outliers.

2. BM25: This is an improvement upon TF-IDF with additional terms to account for long documents.

3. Dense passage retrieval DPR ((Karpukhin et al. 2020): Passages and questions are represented as dense vectors whose representation is obtained by using two separate BERT models. One for encoding passages and other to encode only questions. The ranking functions is in this case the dot product between the query and dense passage vectors. Training DPR involves training the encoders such that dot-product similarity becomes a good ranking function for retrieval. At inference time, the encoder encodes the input question to a d-dimensional vector and this encoding is used to retrieve k passages of which the vectors are the closest to the question vector. Retrieval of passages is done using Approximate Nearest Neighbor Search such as implemented in Facebook AI Similarity Search (FAISS).

Question answering

Question answering is the task of answering questions but abstaining with questions that cannot be answered. It is a discipline within the fields of Natural Language Processing and Information Retrieval.

NLP researchers first developed *closed domain question answering* where the domain is very restricted. The system uses the fact that the domain is smaller to apply simple heuristics to prune out possible answers and extract the most likely value from the domain. *Open domain question answering* is not restricted to any specific domain. These systems can answer any question. In both cases the systems can be made to work with structured data, unstructured data, and semi-structured data.

Currently, the most popular dataset is the Stanford Question Answering Dataset SQuAD. SQuAD belongs to a subdivision of QA known as *extractive question answering,* or *selection based question answering* or *reading comprehension.* This means the answer can be highlighted or selected as part of a document. In contrast, *Generative Question Answering* generates a new answer that is not exactly a quote from the domain. This requires some abstraction power from the model.

Another distinction that is made in question answering systems is between *open book question answering* also called *retrieval based approach* versus *closed book question answering.* Big language models are able to memorize some factual knowledge withing their parameter weights. They can be trained to answer questions without any additional information or context. Such setup enforces the language model to answer questions based on "knowledge" that it has internalized during training. GPT-3 is an example of such model that has been trained on huge amounts of data from the internet.

BERT-like models for question answering

BERT can be adapted for *extractive question answering tasks.* A classification head is put on top of the ouput embeddings of the model, and this classification layer is trained to ouput 2 probability distributions. One for the start-of-answer index and another for the end-of-answer index. These distributions are obtained by taking the Softmax over the output logits of the linear layer. In such a setting, the first sequence presented to the model is the question and the second sequence presented to the model is the passage in

which the answer has to be highlighted. When decoding the output probabilities into an answer, a greedy approach is used where the answer is selected as the subsequence that has the maximum combined start-of-answer and end-of-answer probability. If the answer is inside of the question itself, it is discarded.

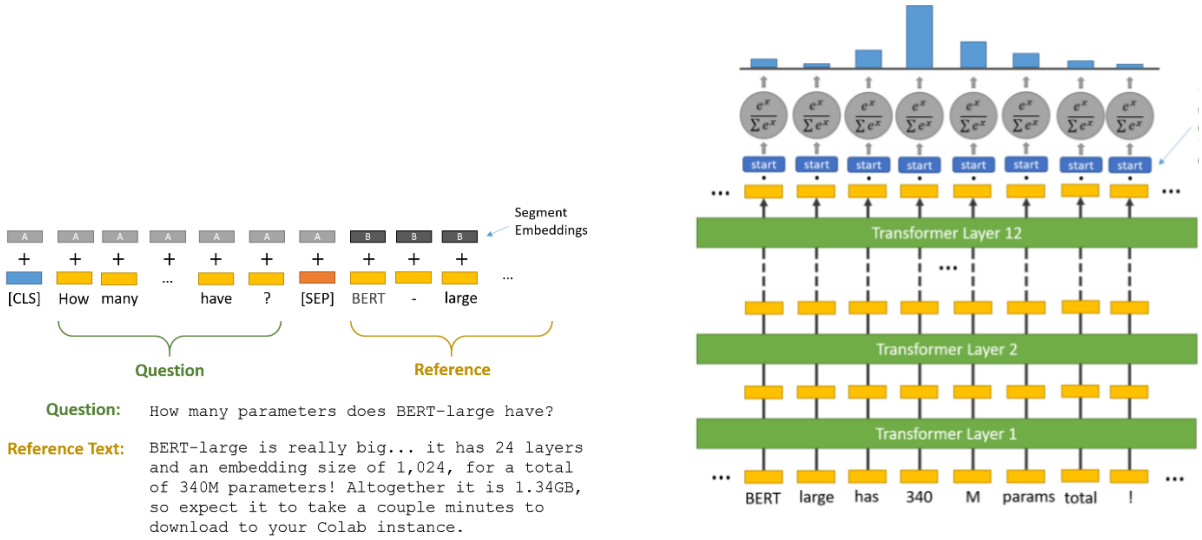


Figure 2.6: When using BERT for question answering, the first input segment is the question, while the second input segment is the paragraph in which the question has to be highlighted. The probability distribution over the paragraph embeddings is used to predict the start of answer token. The same can be done for the end of answer token. This distribution is created by a linear classifier on top of the BERT embeddings with weight sharing. BERT embeddings can be seen as Word2Vec embeddings but with context information, such that every word embedding contains all the information needed to predict whether it is a start- or end-of-answer. (*Question Answering with a Fine-Tuned BERT · Chris McCormick* 2020)

SQuAD

The Stanford Question Answering Dataset is a standard dataset for question answering containing 100,000+ questions posed by crowdworkers on a set of Wikipedia articles. Crowdworkers were asked to create the questions in their own words, without copying content from the passage, and up to 5 questions per passage. The dataset comes in the form of tuples:

$$< passage, question, answer\_start\_index, answer >$$

(Rajpurkar, Jia, and Liang 2018) introduce the concept of "know what you don't know" in the second of version SQuAD2 where questions are added that don't have an answer. Human performance is around 89.45% F1 score which is matched by RoBERTa-large with a 89.80% score. The smaller version RoBERTa-base has an 83% F1 score and is used in our experiments. More recent models outperform humans on SQuAD 1.1 and SQuAD 2.0, which is why researchers are starting to look at more challenging datasets to further push the field.

SQuAD is a restricted version of the real question answering task since the answer can be found as a subset of the document, i.e. a highlight of some portion of a document.

Therefore, there is no need for abstraction or reasoning. There is also no need to understand tables, figures or images. A more accurate name the task of solving SQuAD would be "highlighting correct answers in text".

2.3 Computer vision

Text mining techniques allow to process unstructured textual input, but often forgo the visual clues that humans use for understanding documents. It can therefore be interesting to preprocess layout-intensive documents in order to harness this visual information using computer vision.

2.3.1 Document Layout Analysis

(Xu et al. 2019) cite Document Layout Analysis as one of the early attempts to process documents in an intelligent way for data extraction. Document layout analysis is the process of identifying and categorizing the regions of interest in the scanned image of a text document. After having identified parts of the document as tables, titles, figures or text, one can use a different extraction system for each of the categories. In the context of this thesis, it can be used as a way of detecting tables in order to handle them differently than text.

PubLaynet

Publaynet (Zhong, Tang, and Jimeno Yepes 2019) is the largest available dataset for document layout analysis containing 360,000 document images and covers typical document layout elements such as text, title, list, figure, and table. The difference between this dataset with previous datasets is that they used open access document collection where documents are available in both PDF and XML format, which made it possible to automatically annotate the PDF by matching it with the XML information using the python PDF library PDFMiner. This process allowed them to generate an enormous dataset without any manual work.

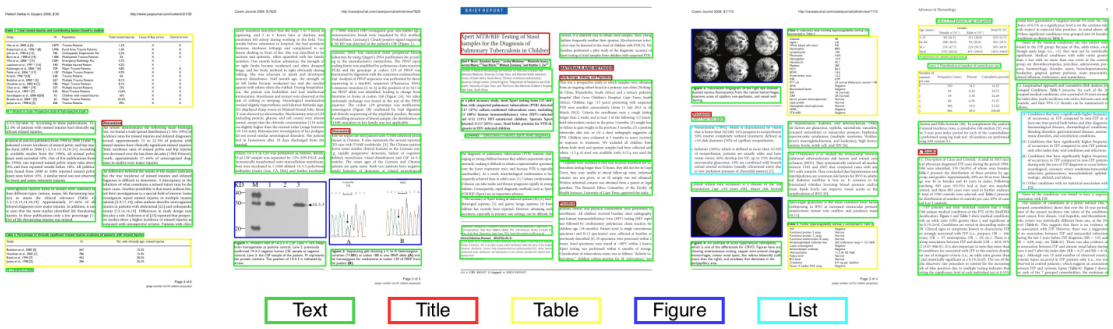


Figure 2.7: Examples from the PubLayNet dataset

Object detection

Faster R-CNN is a popular network used for object detection, and can be used for document layout analysis. It consists of a convolutional backbone network (here Resnet FPN

backbone), a region proposal network and a classification network with bounding box regression. It can also optionally contain a masking network (semantic segmentation) by adding a convolutional network on top of the feature map produced by the Faster R-CNN. This part is not shown in the diagram. This model is provided by Facebook's Detectron2 library and there are trained versions on the PubLayNet dataset available online.

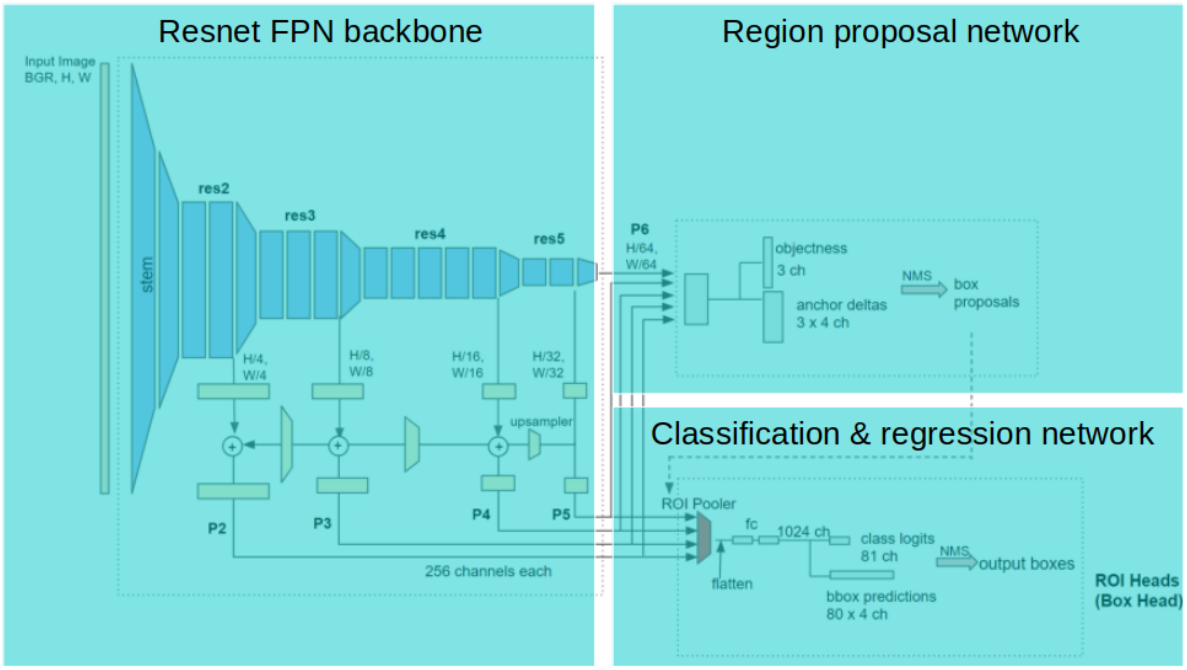


Figure 2.8: Architecture of a Faster-RCNN implementation with ResNet FPN backbone in Detectron2 (Honda 2020). The backbone generates feature maps at different scales. The region proposal network RPN generates bounding boxes of interesting regions. The classification and regression network outputs classes for every region of interest as well as an improved bounding box.

Detectron2 allows to modify the parameters of such a detection network. One particular parameter of interest is the threshold applied to instances after the classification network. This threshold controls the minimum probability required for a Region Of Interest (ROI) classification to be taken into account. It has a direct impact on the precision an recall of the the detected regions and can be optimized for a specific use case.

2.3.2 Table Extraction

Table extraction is a research topic on itself. The task is to locate the bounding boxes of individual cells of a table given an image of this table. It is seen by some as an information extraction task, i.e. labeling words as either cell or a header of some entity-table. It is typically thought of as a 2 stage process: First locate the table (*Table detection*), then extract the cells and headers of the table (*Table recognition or table structure recognition*). Two main approaches are found in the literature: traditional algorithmic methods and more recently methods based on neural networks.

Traditional approach

An example of an traditional approach would be regex parsing. Regex parsing is the use of regular expressions that describe patterns to be matched to drive a search algorithm. However this requires prior knowledge and is very domain specific. Another method is to use morphological operators such as dilation and erosion to detect edges of the table, but this requires having a table with full edges.

Camelot (*Camelot: PDF Table Extraction for Humans — Camelot 0.8.2 documentation* 2021) is a python library for table extraction. It is heavily inspired from Tabula (*Tabula: Extract Tables from PDFs* 2013) and has two extraction options: `Lattice` and `Stream`. The `Lattice` method uses the morphological operators dilation and erosion to detect the lines of a table and then use the intersection of those lines to generate the cell bounding boxes of the table, while the `Stream` method builds rows and columns from the alignment of the words and uses heuristics to build the table.

Deep Learning approach

(Schreiber et al. 2017) propose one of the first deep learning approach by using a Faster-RCNN network for table detection and a Fully Convolutional Network FCN for table structure recognition, they call their approach DeepDeSRT.

In the context of extracting data trapped in unstructured document images such as retail receipts, insurance claim forms and financial invoices in picture format from smartphones,(Paliwal et al. 2020) propose TableNet. TableNet uses a VGG16 backbone followed by one decoder for row segmentation and one decoder for table segmentation. The table is then extracted following a rule-based approach from the detected table and row areas.

Input	Method	Recall	Precision	F1-measure
Image	**DeepDeSRT**	**0.9615**	**0.9740**	**0.9677**
	Tran et al. [16]	0.9636	0.9521	0.9578
PDF	Hao et al. [14]	0.9215	0.9724	0.9463
	Silva [11]	0.9831	0.9292	0.9554
	Nurminen [15]	0.9077	0.9210	0.9143
	Yildiz [34]	0.8530	0.6399	0.7313

Input	Method	Recall	Precision	F1-measure
Images	**DeepDeSRT**	**0.8736**	**0.9593**	**0.9144**
PDFs	Shigarov et al. C_1 [19]	0.9121	0.9180	0.9150
	Shigarov et al. C_2 [19]	0.9233	0.9499	0.9364
	Nurminen [15]	0.9409	0.9512	0.9460
	Silva [11]	0.6401	0.6144	0.6270
	Hsu et al. [15]	0.4811	0.5704	0.5220

Figure 2.9: Benchmark from (Schreiber et al. 2017) comparing different methods on the Marmot table dataset for table detection (left) and table structure recognition (right). Nurminem's approach is used in Camelot and Tabula under the name "Stream".

From accuracies reported above we can see that DeepDeSRT improves on table detection compared to Nurminem, but not on table structure recognition. However DeepDeSRT is not open source, but Tabula and Camelot are, which makes them a good candidate for performing table extraction in this work.

Chapter 3

Implementation

The requirements of the system are to be able to process PDF documents as input, and output string values corresponding the CO2 emissions that were found in this PDF document. In this section we compare three approaches.

The first approach is inspired by table extraction. Tabula and Camelot referenced earlier contain two algorithms for extracting tables. The first one called `Lattice` is based on morphological operators to detect the lines of the table and the second one called `Stream` is based on the alignment of words and uses this to merge words in columns and rows. This stream algorithm inspires an algorithm for extracting CO2 data from documents.

The second approach is based on question answering systems. The standard model used for question answering is BERT and its variants. BERT works very well with text, but cannot handle tabular data, figures and other layout related features. In this second approach we ask ourselves whether it is possible to apply some preprocessing steps to documents in order to make them understandable to BERT. The advantage of BERT is that it is very well documented, many implementations and pretrained models are available, and there are even open source question answering libraries that use this model.

Finally, these two approaches are compared to a layout-aware language model called LayoutLM trained on DocVQA. DocVQA contains around 400 table documents for which questions were generated, and should be able to generalize to some degree to the task of extracting CO2 data.

3.1 Collecting the data: CO2 dataset

In order to evaluate the differnt approaches, a dataset of 30 companies is sampled from the S&P500. The initial idea was to use a scraper to collect this dataset. However it was found that there where many false positives, i.e. sustainability reports of a different company that were returned by the scraper. In other words, the data was not clean enough. The scraper is still in the code of this project as an example of a scraper written with Selenium and using Google Search API to navigate the internet.

Most of these companies release sustainability related documents. The selection criteria is that the CO2 emissions should be reported in a table. This was intentional to make it a hard task for traditional text based question answering systems and study possible solutions. The dataset contains 4 files per company.

- A PDF file of the report

- The single page of the PDF containing the data
- A text file containing the accepted answers.
- An excel file containing the table, extracted by hand.

These documents can be as long as 200 pages. And the information that has to be extracted is reported in a single table on one page. CO2 emissions are typically calculated in different ways. For example, scope 1 emissions relate to direct emissions due to the fabrication of the product that the company sells, scope 2 emissions are the emissions resulting from the energy consumption of the company, and scope 3 emissions are the emissions from the whole life-cycle of the product. These scopes can themselves be computed in different ways and be divided into more detailed categories.

The dataset is partitioned into 20 documents for training (i.e. choosing the right parameters of the system) and 10 documents for testing purposes.

3.2 Grid algorithm

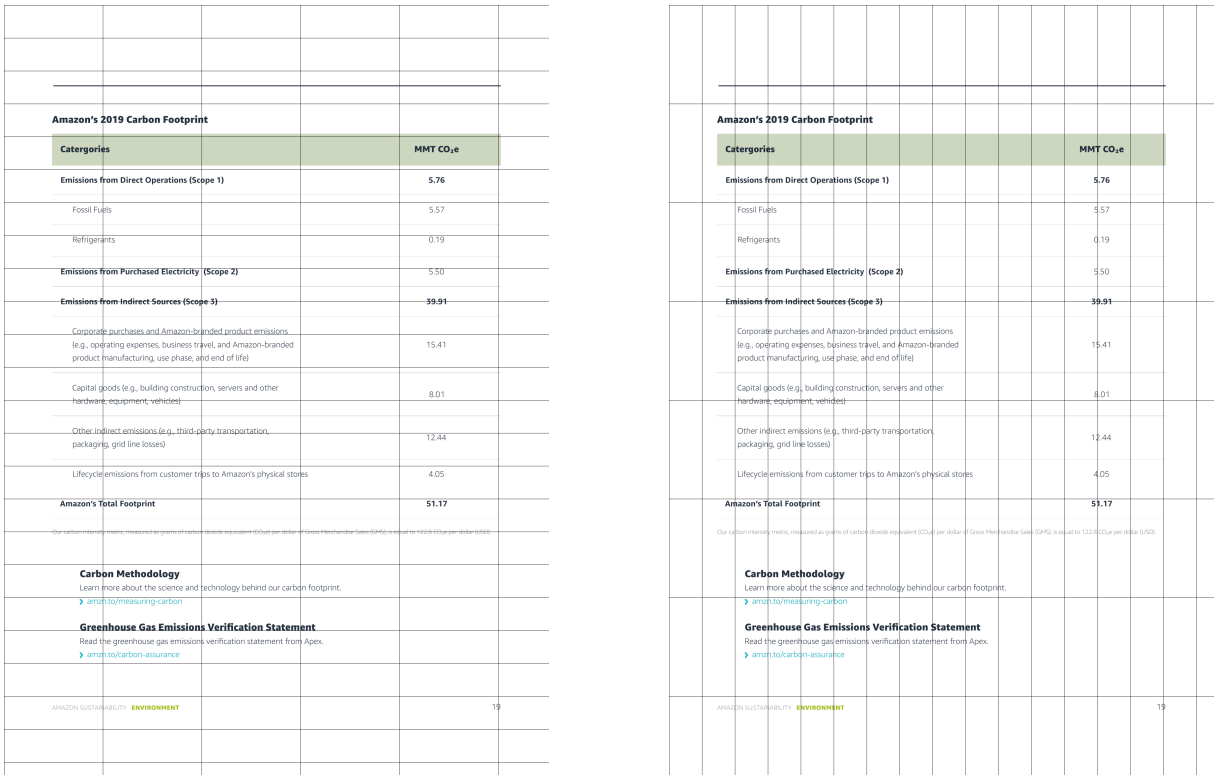
Camelot and Tabula are table extraction software that have a table extraction method called `Stream` based on (Elomaa 2013). The essence of the method is to detect table areas by looking for parts of the PDF that have a lot of words aligned vertically and then combining them into a table.

Inspired by this method, we propose to segment PDF pages into grids. Every intersection of lines is considered to be a bucket in which words can be partitioned. Candidates are then extracted when they are on the same row/column as a keyword and under a certain distance threshold. The size of the cells can be optimized. A filtering step is added after finding the candidates to remove obviously wrong answers. This includes removing numbering like 1. 2. 3. or removing numbers between 2000 and 2025.

Layout analysis is performed by the PDF parser PDFMiner (*pdfminer · PyPI* 2019) that outputs a list of paragraphs based on the distance between characters. When a paragraph is too long, it is discarded as it is considered to be text and not a potential table entry. This is necessary because PDF was never designed as a data input format, but as an output format. The aim of PDF was to be highly customizable. At its core, PDF consists of a set of instructions that describe how to draw the page. Data isn't stored in words, sentences or paragraphs but as characters which are painted at certain locations on the page. As a result, most semantics are lost when a text or word document is converted to PDF. All structures are converted to a list of characters floating on the pages. PDF readers do a good job at reconstructing this structure based on the distances between the characters, but it is not perfect. Some PDF encoders add white spaces between characters, others remove all white spaces for compression, and the layout analysis performed by the reader is not perfect. So, in practice, the PDF reader partitions the text into blocks which it believes to be individual entities in the document, but sometimes it makes mistakes such as splitting a paragraph in two or adding white spaces between the characters of a word (although this is rare).

```
def get_grids(PDF, row_tol, col_tol, paragraph_threshold):
""" returns a PDF segmented as a grid"""
    for page in PDF:
        width, height = 595, 842
        rows = list(0, 1*row_tol, 2*row_tol, ..., height)
        columns = list(0, 1*col_tol, 2*col_tol, ..., width)
        grid = Grid(rows, columns)
        for paragraph in page:
            if len(paragraph) > paragraph_threshold:
                continue
            for word in paragraph:
                grid.put(word)
        yield grid

def Filter(w):
    """ filter out some digits"""
    if not w.digit() or w.endwith(".") or w.between(2000,2025):
        return False
    return True

def get_digits(line, keywords, thresh):
""" returns digits contained in the string """
    for word in line:
        for keyword in keywords:
            if(dist(word, keyword)<thresh) and Filter(word):
                yield word

def get_candidates(grids, keywords, thresh):
""" return digits found in the grid on the same row/column as
a keyword withing a threshold distance"""
    candidates = []
        for grid in grids:
            for row in grid.rows:
                c = get_digits(row, keywords, thresh)
                candidates.extend(c)
            for column in grid.columns:
                c = get_digits(column, keywords, thresh)
                candidates.extend(c)
    return set(candidates)

grids = get_grids(PDF, 30, 30)
keywords = ["co2", "emission", "scope", "ghg"]
get_candidates(grids, keywords, 400)
```

Figure 3.1: Pseudo-python code of the GRID algorithm. A PDF is partitioned into a grid of buckets. Paragraphs are discarded if they are too long. Candidates are considered if they appear on the same row/column under a certain threshold distance from a keyword.



Amazon's 2019 Carbon Footprint

Catergories	MMT CO_2e
Emissions from Direct Operations (Scope 1)	**5.76**
Fossil Fuels	5.57
Refrigerants	0.19
Emissions from Purchased Electricity (Scope 2)	**9.50**
Emissions from Indirect Sources (Scope 3)	**39.91**
Corporate purchases and Amazon-branded product emissions (e.g., operating expenses, business travel, and Amazon-branded product manufacturing, use phase, and end of life)	15.41
Capital goods (e.g., building construction, servers and other hardware, equipment, vehicles)	8.01
Other indirect emissions (e.g., third-party transportation, packaging, grid line losses)	12.44
Lifecycle emissions from customer trips to Amazon's physical stores	4.05
Amazon's Total Footprint	**51.17**

Our carbon intensity metric, measured as grams of carbon dioxide equivalent (CO_2e) per dollar of Gross Merchandise Sales (GMS), is equal to 122.8 CO_2e per dollar (USD).

Carbon Methodology
Learn more about the science and technology behind our carbon footprint.
› amzn.to/measuring-carbon

Greenhouse Gas Emissions Verification Statement
Read the greenhouse gas emissions verification statement from Apex.
› amzn.to/carbon-assurance

AMAZON SUSTAINABILITY ENVIRONMENT 19

Amazon's 2019 Carbon Footprint

Catergories	MMT CO_2e
Emissions from Direct Operations (Scope 1)	**5.76**
Fossil Fuels	5.57
Refrigerants	0.19
Emissions from Purchased Electricity (Scope 2)	**9.50**
Emissions from Indirect Sources (Scope 3)	**39.91**
Corporate purchases and Amazon-branded product emissions (e.g., operating expenses, business travel, and Amazon-branded product manufacturing, use phase, and end of life)	15.41
Capital goods (e.g., building construction, servers and other hardware, equipment, vehicles)	8.01
Other indirect emissions (e.g., third-party transportation, packaging, grid line losses)	12.44
Lifecycle emissions from customer trips to Amazon's physical stores	4.05
Amazon's Total Footprint	**51.17**

Our carbon intensity metric, measured as grams of carbon dioxide equivalent (CO_2e) per dollar of Gross Merchandise Sales (GMS), is equal to 122.8 CO_2e per dollar (USD).

Carbon Methodology
Learn more about the science and technology behind our carbon footprint.
› amzn.to/measuring-carbon

Greenhouse Gas Emissions Verification Statement
Read the greenhouse gas emissions verification statement from Apex.
› amzn.to/carbon-assurance

AMAZON SUSTAINABILITY ENVIRONMENT 19

Figure 3.2: Different grids applied to a sustainability report of Amazon. The width and height of cells in the grid can be modified. Every intersection of lines is a bucket that partitions the document. The distance unit of the PDF coordinate system is 1/72 inch. This means that an A4 report has the dimensions width x height = 595 x 842.

3.3 RoBERTa with document layout analysis

This method evaluates how Document Layout Analysis and table extraction could be used as a preprocessing step to a question answering system based on RoBERTa. The first step is thus to detect tables and text in the document. For this we use Detectron2 (*facebookresearch/detectron2* 2021), a framework developed by Facebook that contains many implementations for computer vision. They provide YAML configuration file system to tweak the parameters of the network and there are many pretrained networks available online. The implementation used in this work is a Mask R-CNN with FPN Resnet backbone (*hpanwar08/detectron2: Detectron2 for Document Layout Analysis* 2021) pretrained on PubLayNet that yields bounding boxes with the labels text, list, figure, table and title. The PDF is thus preprocessed in order to handle text and tables separately. Text can be used without further processing. Tables are handled separately by a dedicated table extraction software Camelot (*Camelot: PDF Table Extraction for Humans — Camelot 0.8.2 documentation* 2021). Camelot is a python table extraction library inspired by Tabula and based on PDFMiner. It is used because it is a good balance between speed and accuracy. As discussed earlier, deep learning methods for table extraction are the SOTA solution. But adding another computationally expensive component to the system seemed unwise. Tables are then translated into text and fed into the question answering system based implemented using Haystack. Haystack (*Haystack - Question Answering* 2021) is a python library that provides a question answering pipeline with shallow retrievers (index based) and dense readers (neural based). The hypothesis that is tested is whether we can

find textual representation of the table that will be understood by the Retriever and the Reader in order to adapt traditional language based question answering system to also work with tables in documents.

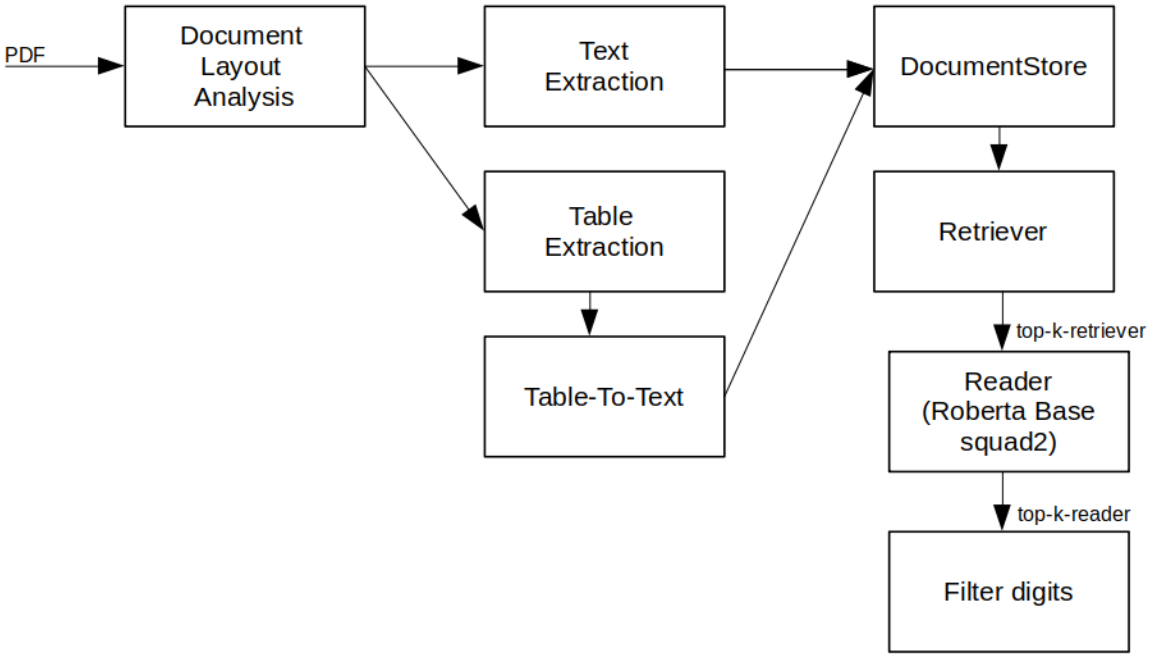


Figure 3.3: Document layout analysis followed by table extraction and table to text conversion which is then introduced into question answering system whose results are filtered to keep only digits. The reader uses a RoBERTa-base model trained on SQuAD2 and the retriever is based on TF-IDF.

3.3.1 Table-to-text

Tables can have very complex structures. We simplify this definition as shown in the following figure. This means that we will automatically consider the first row and the first column be headers. If sub-headers are present in the table, they will be counted as a normal cell.

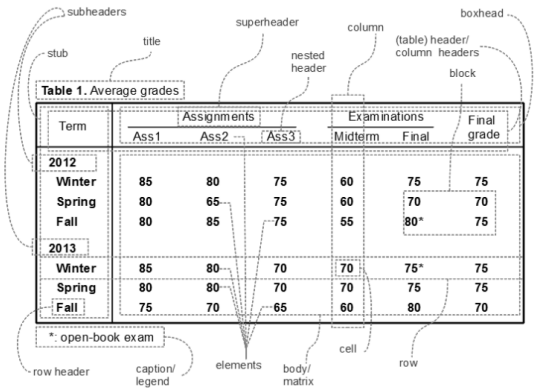


Figure 3.4: Model of a table as defined in (Elomaa 2013)

Main Header	Column Header	Column Header	Column Header	Column Header
Row Header	cell	cell	cell	cell
Row Header	cell	cell	cell	cell
Row Header	cell	cell	cell	cell

Figure 3.5: Simplified model of a table with row headers, column header and one main header.

Tables are transformed to a textual representation. Different ways of doing this are tried in the experiments section, but the main idea is to create a sentence for every cell of the table in the following way:

Main Header and **Row Header** have value **cell** for **Column Header**

The CO2 dataset is built with this idea in mind. Therefore, for every PDF, a table is included. This table has been interpreted from the PDF document into the simplified structure. It is made sure that the headers contain all the information necessary to correctly identify the cell as being a CO2 value, and will allow to test how well the question answering system works on "perfect" tables in the experiments.

In the real setting with the Document Layout Analysis, every table will be extracted and converted to text as if it had this structure, which leads to some errors, and this is evaluated in the experiments section.

3.4 LayoutLM

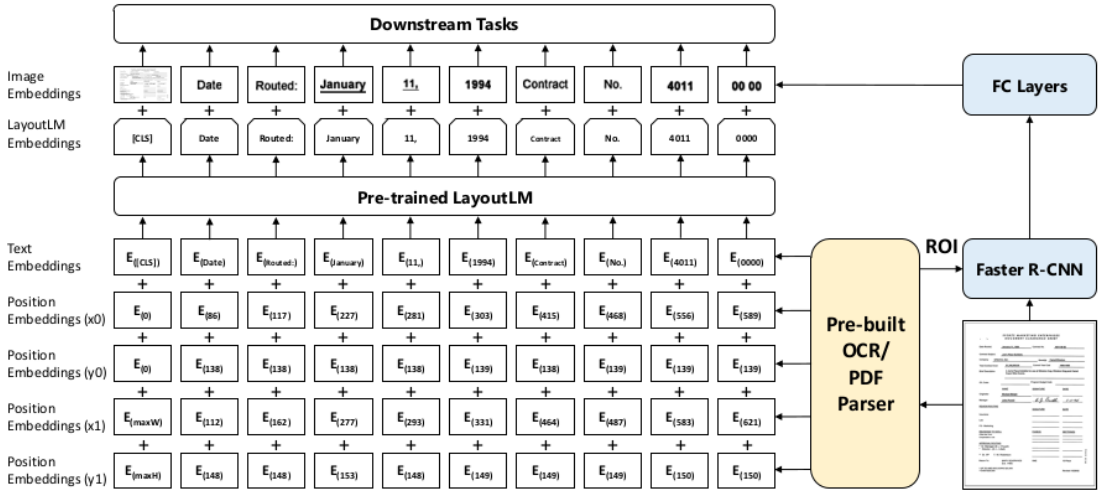


Figure 3.6: LayoutLM (based on BERT) and Faster R-CNN generating embeddings for downstream tasks

LayoutLM adapts the BERT architecture for 2D layout understanding. They also jointly train the model with image embeddings generated by a Faster R-CNN. A quote from their paper explains the concept:

> *"Unlike the position embedding that models the word position in a sequence, 2-D position embedding aims to model the relative spatial position in a document. To represent the spatial position of elements in scanned document images, we consider a document page as a coordinate system with the top-left origin. In this setting, the bounding box can be precisely defined by* (x_0, y_0, x_1, y_1)*, where* (x_0, y_0) *corresponds to the position of the upper left in the bounding box, and* (x_1, y_1) *represents the position of the lower right. We add four position embedding layers with two embedding tables, where the embedding layers representing the same dimension share the same embedding table. This means that we look up the position embedding of* x_0 *and* x_1 *in the embedding table X and lookup* y_0 *and* y_1 *in table Y"*

The team used a BERT implementation from the Transformers library (*Transformers Huggingface* 2021). There is an option to bypass the model's internal embedding lookup matrix and provide the embeddings directly to the model. They use this to provide their custom 2D embeddings. A look at the source code shows that in addition to the X and Y embeddings, they use width and height embeddings. All the embeddings are summed. LayoutLM is very similar to BERT and the implementation holds in less than 300 lines of Python.

```
word_embeddings = Embedding(30522, 768)
position_embeddings = Embedding(512, 768)
x_position_embeddings = Embedding(1024, 768)
y_position_embeddings = Embedding(1024, 768)
h_position_embeddings = Embedding(1024, 768)
w_position_embeddings = Embedding(1024, 768)
token_type_embeddings = Embedding(2, 768)

embeddings = (
            words_embeddings(word_ids)
            + position_embeddings(position_ids)
            + x_position_embeddings(x0)
            + y_position_embeddings(y0)
            + x_position_embeddings(x1)
            + y_position_embeddings(y1)
            + h_position_embeddings(y1-y0)
            + w_position_embeddings(x1-x0)
            + token_type_embeddings(token_type_ids)
        )
```

Figure 3.7: Python source code of LayoutLM creating custom embeddings that are provided to a BERT model. "Embedding" refers to torch.nn.Embedding in the pytorch library. Note that they use traditional learned embeddings for the 1D position embeddings instead of the sin/cos embedding used in the Transformer and in the standard BERT implementations.

(*anisha2102/docvqa: Document Visual Question Answering* 2021) provide a trained version of LayoutLM on the DocVQA dataset. In their training however, they did not

use the joint training with image embeddings from a Faster-RCNN. They achieve 68.93% exact match score on the DocVQA dataset. This trained model is used to compare against on the CO2 dataset.

3.4.1 Adapting the implementation to PDF documents and CO2 extraction

BERT for question answering predicts the probability distribution of the start- and end-of-answer tokens, which is decoded in a greedy way to predict a single answer per input. This means that a single answer is predicted per paragraph given to the model. Their implementation is modified to iterate over a full PDF and only keep the 10 most likely candidates. This is done by taking the cross product of the start logits with the end logits. The 10 combinations with largest combined probability are kept. A final filtering step is applied to only keep digit values from the provided answers.

Chapter 4

Experiments

Three approaches are considered: GRID algorithm, a RoBERTa based question answering sytem with Document Layout Analysis as preprocessing and finally a layout aware language model called LayoutLM.

4.1 Evaluation

For each document, multiple CO2 emissions values have to be extracted. It thus makes sense to look at the precision and recall values, as well as the F1 metric. All the three implementations output only digits, so precision and recall are computed by looking at the number of exact matches between the method and the reference answers from the dataset.

$$precision = \frac{\text{\# correctly extracted CO2 emissions numbers}}{\text{\# total number of extracted CO2 emission numbers}}$$

$$recall = \frac{\text{\# correctly extracted CO2 emissions numbers}}{\text{\# total correct CO2 emissions numbers}}$$

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

In addition to these metrics we use another metric that we call "Found One correct". This metric gives the percentage of documents for which at least one correct CO2 value was retrieved. Some results show very low accuracies for F1, precision and recall. So Found One Correct can give insights into the amount of PDFs for which the extraction process completely failed.

4.2 Grid algorithm

The GRID algorithm partitions the PDF into buckets in order to extract values along rows and columns of predefined keywords. The size of the grid is tuned in this section. The PDF layout analysis performed by the PDF parser outputs blocks of text, which are discarded if they are too long. The length to discard text is also tuned in this section. Finally the impact of different keyword on accuracy is measured. Experiments are performed on the single page version of the PDF in the dataset.

4.2.1 Parameters used in the experiments

```
ROW_TOL = 100
COL_TOL = 100
CANDIDATE_THRESHOLD = 400
PARAGRAPH_THRESHOLD = 100
KEYWORDS = ["co", "scope", "co2", "emission", "ghg"]
```

Figure 4.1: ROW_TOL and COL_TOL control the distance between lines of thegrid. CANDIDATE_THRESHOLD is the maximum distance allowed between a keyword and a candidate. PARAGRAPH_THRESHOLD is the amount of characters above which a textbox is considered to be text, and not a potential entry of a grid.

4.2.2 What is the best grid size?

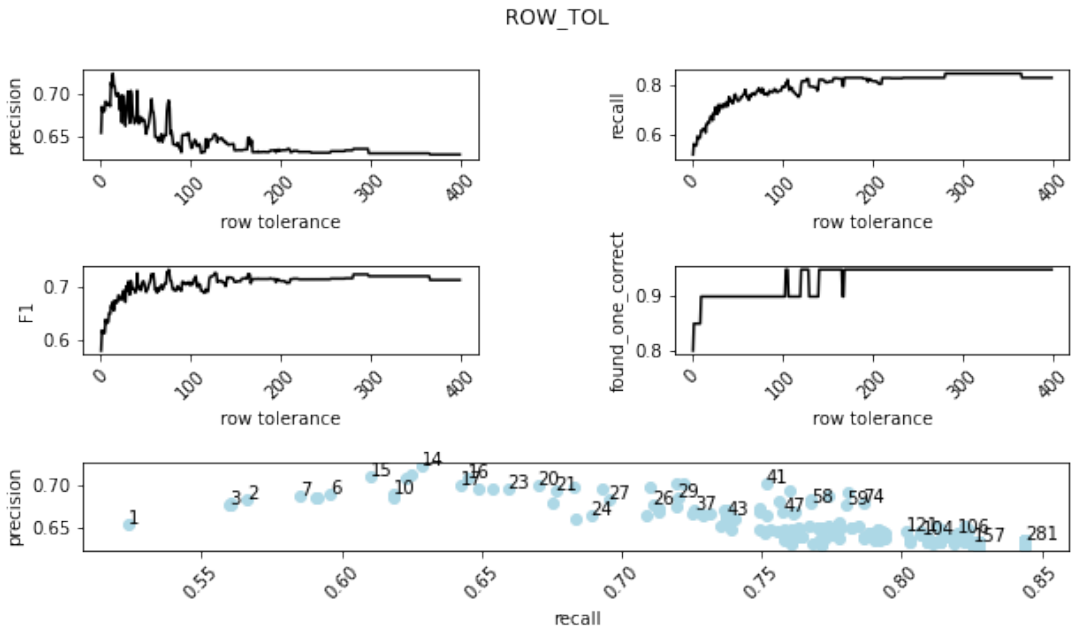


Figure 4.2: Precision drops with increasing row size, but recall increases. Overall the increase in recall dominates which is why the F1 score goes up.

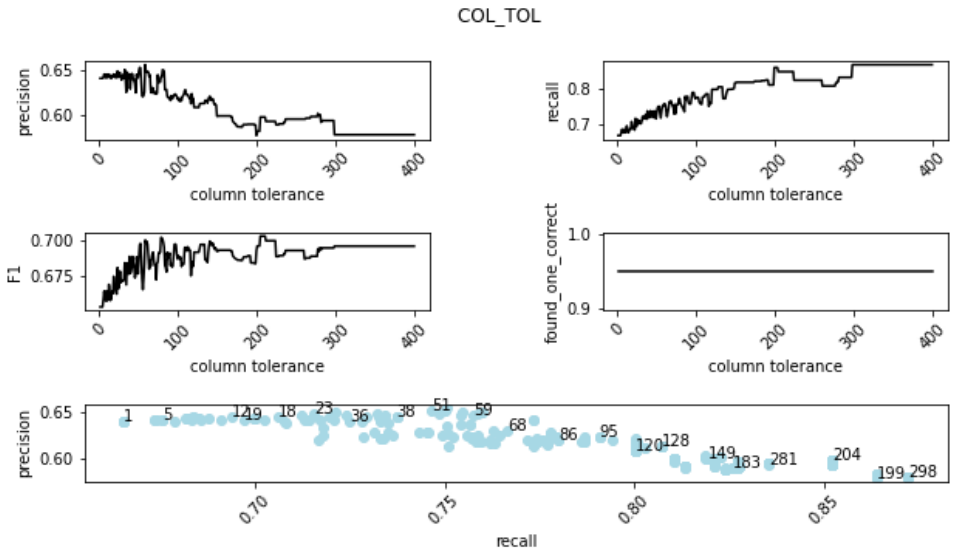


Figure 4.3: Again, precision drops with when the column tolerance increases, which means wider columns, while the recall increases. Overall the increase in recall dominates which is why the F1 score increases. The curve is noisy and can be explained by the small dataset.

One important result to note from this is that although precision drops when creating a coarser grid, recall increases more which causes the F1 score to increase. But even so, the drop in precision is small. The system still performs well with very big values for row height and column width (around 400) which corresponds to creating one big bucket per PDF page. The improvement of using a grid compared to one big bucket per page is not significant.

4.2.3 What is the best distance threshold?

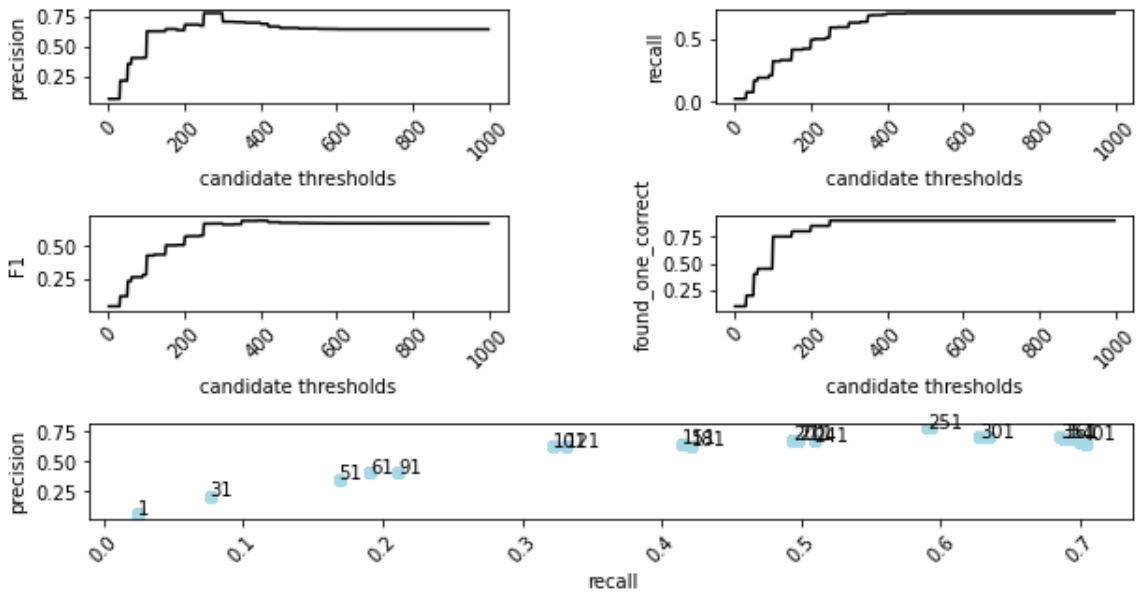
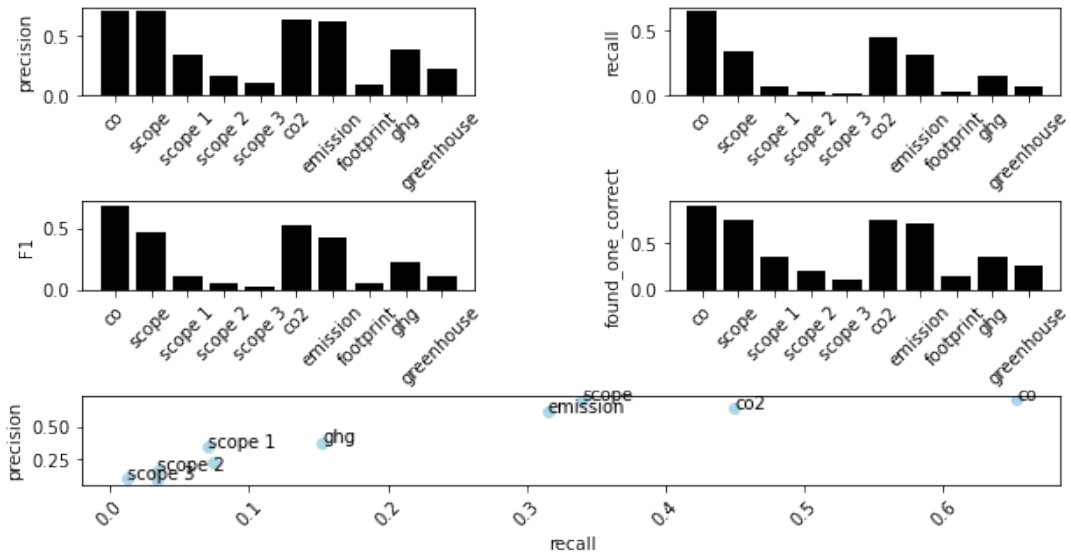


Figure 4.4: The metric used to rank candidates is the distance between the candidate and the keyword on the PDF. The experiments show that a value of 400 is optimal, which is around the width of the PDF.

4.2.4 What are the best keywords?



In this experiment, the extraction is ran with every individual keyword to measure its impact. The best keywords are to “co”, “scope”, “co2”, “emission” and “ghg”. Scope 1, Scope 2 and Scope 3 are removed from the keywords as they are redundant with the Scope keyword. Footprint is removed because it has a very low impact. “Greenhouse” is removed because it was noted that in practice it almost always appears in combination with the abbreviation “ghg”.

4.2.5 Test set

Knowing that all the information is stored in tables in the documents, the straightforward approach would be to extract the tables, and then extract numbers the same way as with the GRID algorithm. So instead of building the grid on the full page, building it using table extraction. On the test set we compare the two approaches, which give similar results.

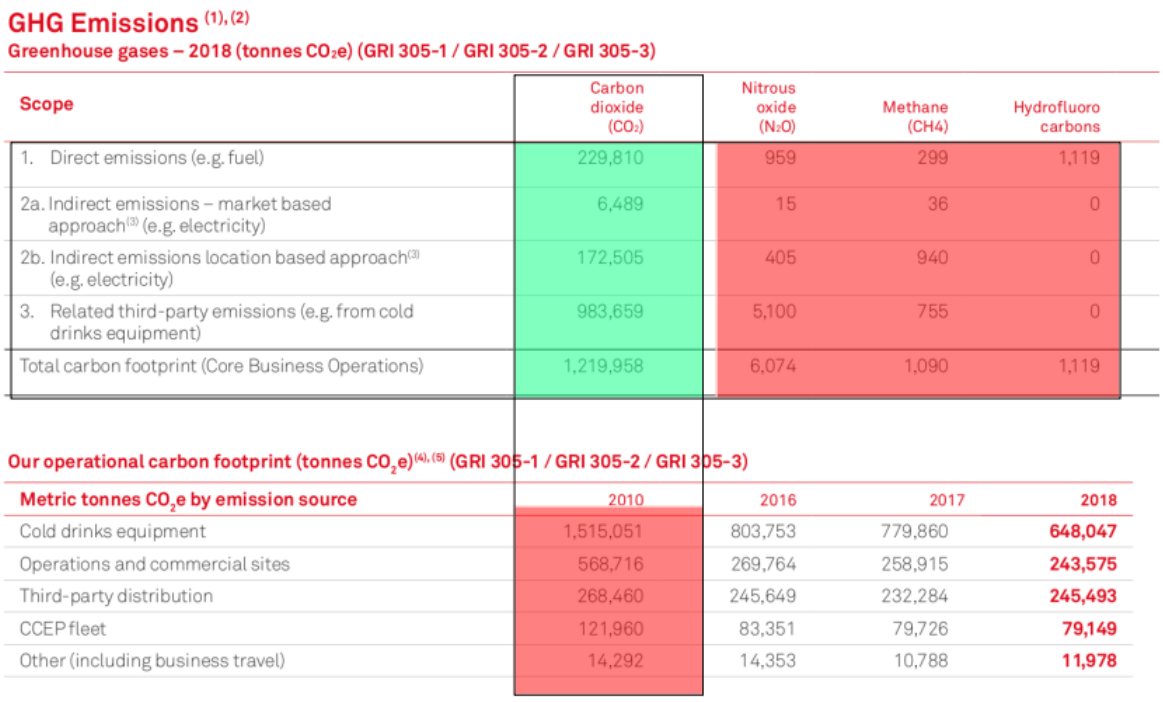
type	found one correct	precision	recall	F1
GRID on single page PDF	0.8	0.44	0.62	0.51
Table extraction on single page PDF	0.7	0.41	0.51	0.46
GRID on multiple pages PDF	1.0	0.24	0.74	0.36
Table extraction on multiple pages PDF	0.9	0.28	0.64	0.39

Precision drops when using the full PDF version and recall increases, but the overall change causes a drop in F1 score. It may seem strange that "Found one correct" increases by using the full PDF version, as no answers should be found in the added pages, but this is not true. The emissions are often discussed in the text of the PDF document which can be on a different page, and explains the increase in this metric.

4.2.6 Commonly encountered problems

ENVIRONMENTAL	2020
Energy Consumption (MWh)[1,2]	3,090,000
% Renewable Electricity[1,2]	32%
Scope 1 emissions (metric tons CO_2e)[1,2]	333,000
Scope 2 (market-based) emissions (metric tons CO_2e)[1,2]	433,000
Water Consumption (megaliters)[2]	20,246
% of Waste Recycled[2]	78%
Total Waste (metric tons)[2]	88,222
Hazardous Waste (metric tons)[2]	7,999
# of ISO14001 manufacturing sites certified[3]	13

Figure 4.5: Extraction for the sustainability report of Deere corp completely fails. Because the characters are very close, the PDF parser interprets the left column as one big chunk of text which causes the GRID algorithm to discard the text.



GHG Emissions (1), (2)

Greenhouse gases – 2018 (tonnes CO_2e) (GRI 305-1 / GRI 305-2 / GRI 305-3)

Scope	Carbon dioxide (CO_2)	Nitrous oxide (N_2O)	Methane (CH_4)	Hydrofluoro carbons
1. Direct emissions (e.g. fuel)	229,810	959	299	1,119
2a. Indirect emissions – market based approach(3) (e.g. electricity)	6,489	15	36	0
2b. Indirect emissions location based approach(3) (e.g. electricity)	172,505	405	940	0
3. Related third-party emissions (e.g. from cold drinks equipment)	983,659	5,100	755	0
Total carbon footprint (Core Business Operations)	1,219,958	6,074	1,090	1,119

Our operational carbon footprint (tonnes CO_2e)(4), (5) (GRI 305-1 / GRI 305-2 / GRI 305-3)

Metric tonnes CO_2e by emission source	2010	2016	2017	2018
Cold drinks equipment	1,515,051	803,753	779,860	**648,047**
Operations and commercial sites	568,716	269,764	258,915	**243,575**
Third-party distribution	268,460	245,649	232,284	**245,493**
CCEP fleet	121,960	83,351	79,726	**79,149**
Other (including business travel)	14,292	14,353	10,788	**11,978**

Figure 4.6: Extracting the values for a sustainability report of Coca-Cola gives a precision of 1 but a recall of 0.13. This can in part be explained by the red parts in the figure that also get extracted by the GRID algorithm.

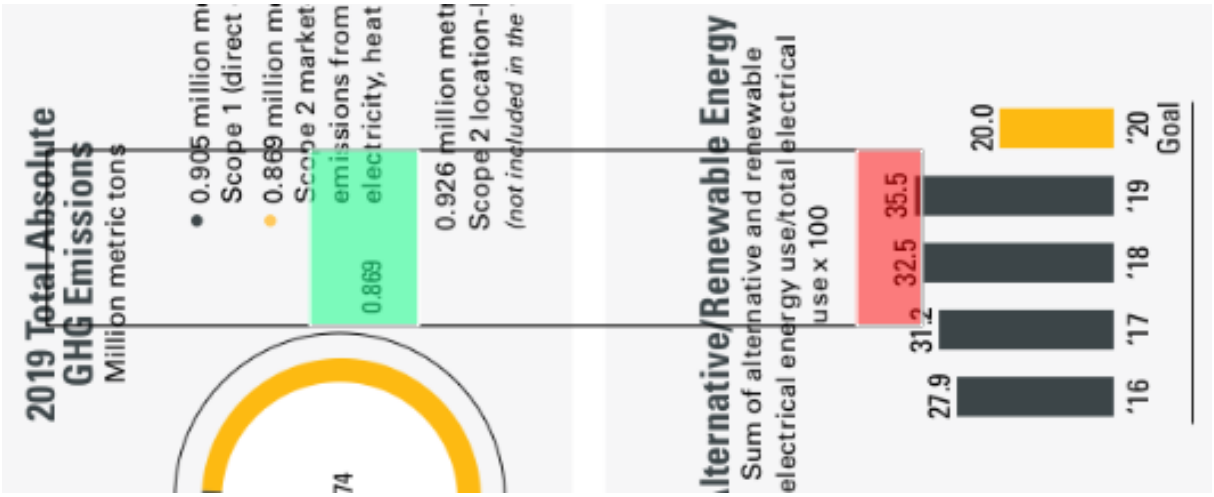


Figure 4.7: This report from Caterpillar shows data in the form of figures. When interpreting this as a grid, values from different figures are incorrectly extracted.

Key Performance Indicator	Data
Energy consumption within the organization	Fuel consumption: 2,883,506,400,000,000 joules Electricity consumption: 1,074,586 MWh
	Total scope 1 & 2: 1,875,560 MWh
Energy intensity	0.0332275 MWh/Sq Ft
Reduction of energy consumption	34,415 MWh or 1.8% reduction year-over-year
Direct GHG emissions (Scope 1)	219,869 MT CO_2e
Energy indirect GHG emissions (Scope 2)	*354,451 MT CO_2e
Other indirect GHG emissions (Scope 3)	1,019,791 MT CO_2e
GHG emissions intensity	0.01017468 MT CO_2e/Sq Ft
Reduction of GHG emissions	*120,444 MT CO_2e, or 17.3% reduction year-over-year

Figure 4.8: In this report from Bank Of America, on can see that the system extracts the value corresponding to a reduction of GHG emissions, which is not an accepted answer. This is because the system has no understanding of language.

4.2.7 Discussion and improvements

The GRID algorithm is a straightforward way to extract numerical values from a PDF given a list of relevant keywords. In practice, it is observed that a very coarse grid works well on the CO2 dataset because the increase in recall dominates the decrease in accuracy. Overall, the accuracy is similar than if one were to use table extraction software followed by digit extraction in the table.

Low precision is caused by a lack of context and language understanding. For example, no difference is made between a year and an emission value, or the difference between an absolute value and a reduction in emission. The following steps could be taken to improve precision:

- add a blacklist of keywords
- require a unit such as metric tons
- improve filtering rules

The biggest flaw of this method is that the partitioning of the PDF document happens in a rather arbitrary way, without looking at the individual page. The partitioning of the page could be done in a smarter way by looking at the alignment of words. Rossum.ai for example plots a histogram of the image of the PDF, in order to detect alignments in the image. The same idea is used in the STREAM algorithm in Tabula and Camelot, which compute the columns by looking at the coordinates for which there are a lot of vertically aligned words.

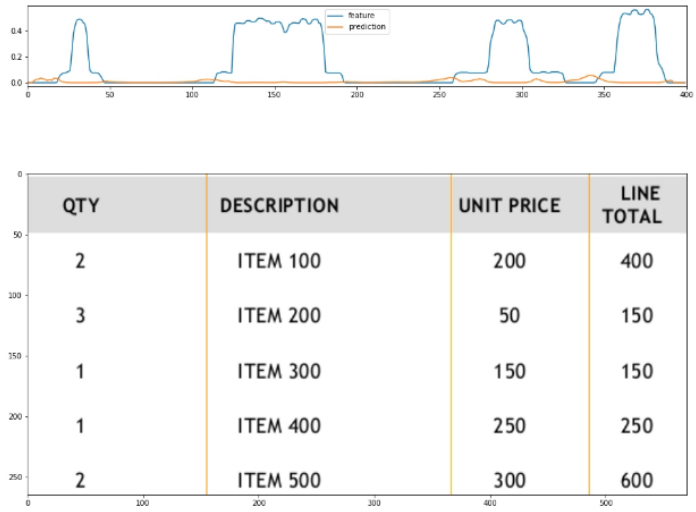


Figure 4.9: Rossum.ai's method to detect separation lines in a table using the histogram of pixels. The horizontal histogram of the intensity of the inverted image is a good indicator of vertical alignment of words in the image and can be used to detect the separation lines of those columns. This method could be implemented to detect an adaptive grid for partitioning the PDF document. (*Rossum's line item extraction from invoices* 2018)

This rule-based approach can very quickly grow to become complex, and is a fight between accuracy and precision. Some rules might work well on some PDFs but not on others. Furthermore, this approach would be suited only numerical data extraction. What if one wanted to extract from a report the answer to:"What is the number one goal of the company in the coming years ?", or "What is the reduction in emissions since 2010 ?". This would require a complete redesign of the system.

4.3 RoBERTa with DLA preprocessing

The second approach is to apply Document Layout Analysis as a preprocessing step for standard question answering systems, in order to transform tables into a textual representation that preserves layout. In the following, we present different experiments to tweak the parameters of this system and evaluate their impact.

4.3.1 Parameters of the experiments

```
USE_GPU = True
TOP_K_RETRIEVER = 20
TOP_K_READER = 20
QUERY = "What are the total co2 emissions in metric tons?"
TRANSLATION = "row_first"
THRESHOLD_ACCURACY = 0.3
BOX_THRESHOLD = 0.1
```

Figure 4.10: Experiments are performed using an NVIDIA GeForce GTX 1060. The number of outputs of the retriever (TF-IDF) and the reader(roberta-base-squad2 available as a pretrained model on the python Transformers library (*Transformers Huggingface* 2021)) are set to 20. The query is also fixed. The translation from table to text is selected as the one explained in the implementation chapter. In the experiments we test other translations. Answers returned by the question answering system are thresholded to have a minimum probability of 0.3. The bounding box threshold accuracy from the document layout analysis step is set to 0.1, which is very low to allow as many detections as possible. Because accuracies are low, and because of computational cost, the experiments are performed on the single-page version of the PDF unless specified otherwise.

4.3.2 Finetuning on synthetic data

Initial experiments with RoBERTa show that the model has very low output probabilities, which indicates that the model is very unsure. Precision and recall is also very low. Therefore, the model is finetuned on synthetically generated data.

First, a big text corpus is generated from the CO2 dataset. Then two paragraphs are randomly sampled from this corpus. The training example is generated by concatenating the first paragraph, then a sentence of the form “Main header” and “row header” have value “cell value” for “column header” with a randomly sampled cell value and Main header and row header randomly sampled out of a list of keywords. Then the last paragraph is added at the end.

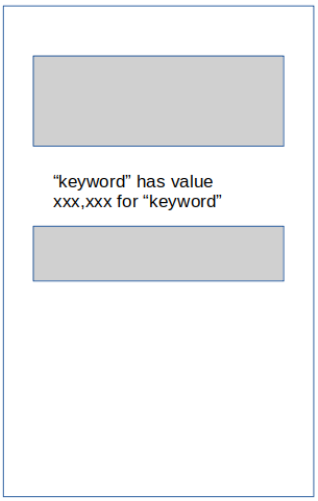


Figure 4.11: example of a synthetically generated example. The two grey boxes are randomly sampled text from a text corpus generated out of the training set. A sentence representation of a table cell is inserted in between, where “keyword” is sampled from [”emissions”, ”co2”, ”ghg”, ”footprint”, ”scope 1”, ”scope 2”, ”scope 3”, ”greehouse gas emissions”] and XXX,XXX is replaced by randomly sampled numbers.

type (single page PDF)	threshold	found on correct	precision	recall	F1
RoBERTa pretrained on SQuAD2	0.3	0.05	0.05	0.006	0.01
RoBERTa pretrained on SQuAD2	0.0	0.45	0.39	0.06	0.11
RoBERTa finetuned on Synthetic	0.3	0.7	0.55	0.13	0.21

Table 4.1: The results show that finetuning the model on synthetically generated data increases the confidence of the model by increasing its output probabilities. It also results in a slight improvement of the F1 score.

4.3.3 What is the impact of the DLA step ?

The document layout analysis has two purposes. First, it helps to detect tables, which are later extracted by dedicated software. Second, it prevents paragraphs from being cut in two arbitrarily pieces before being fed into the question answering system. It preserves the initial segmentation of the text into different paragraphs, and should prevent an answer from being split in to. This second purpose should however have no implications on performance compared to a standard BERT model since it uses a sliding-window approach to solve this issue. So the real benefit here is the table extraction part.

Figure 4.12: Visualization of the document layout analysis that correctly classifies different regions as either text, title, figure or table.

type (single page PDF)	found one correct	precision	recall	F1
PDF	0.15	0.15	0.06	0.09
DLA with table extraction	0.75	0.53	0.12	0.20
human extracted tables as input	0.9	0.78	0.18	0.28

The accuracy obtained by flattening the PDF into a text is extremely low. However, it can be increased by using DLA and table extraction. This would be expected since the data that is being extracted is always located in tables. The fact that the system without DLA is still able to find correct answers can be explained by the fact that emission values

in tables are often discussed in the text of a document, making it accessible to standard textual questions answering system and that many tables have a horizontal interpretation so that they can be read as text. It is however surprising that a model that achieves near human performance on the SQuAD2 dataset yields such low accuracies for this task.

4.3.4 What is the best DLA output threshold ?

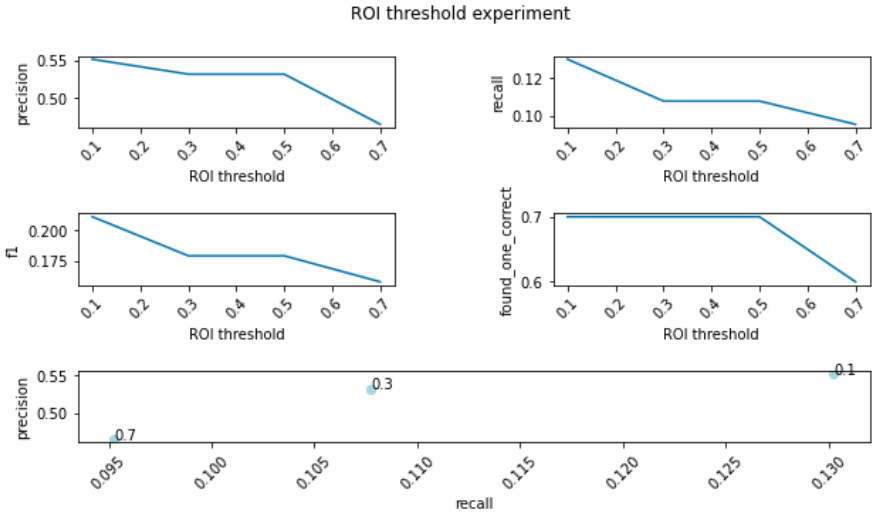
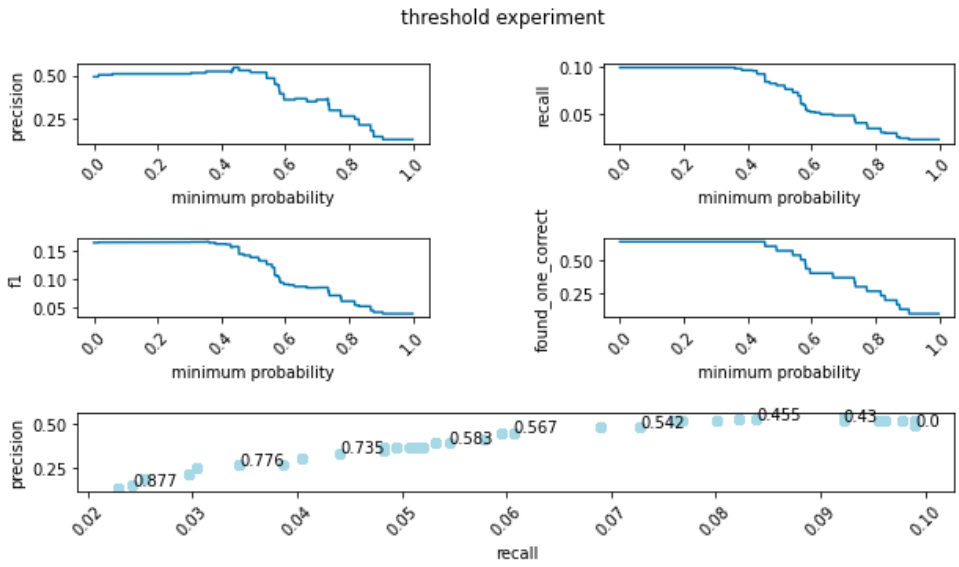


Figure 4.13: The threshold applied on classifications at the output of the object detection network.

The lower the threshold for a region of interest during document layout analysis, the smaller the probability that the system is going to fail to extract the information from the PDF. This explains the decrease in recall seen when increasing the threshold. Accuracy also decreases, which might be explained by the fact that correct answers get discarded when the threshold probability increases. These results should however be taken with criticism because the dataset is very small (20 documents) and the decrease in recall/precision is mild.

4.3.5 What is the best threshold probability?

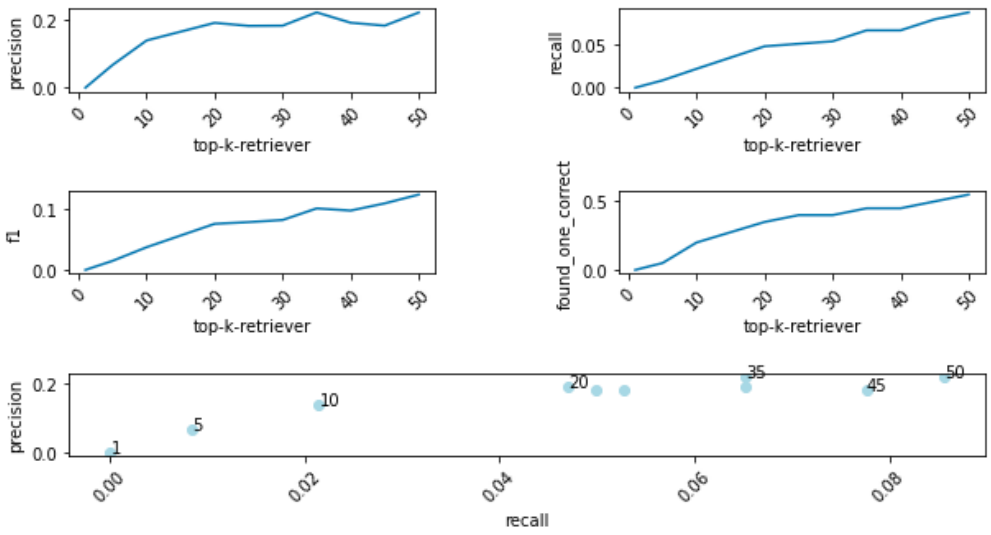
The RoBERTa model used for question answering outputs probabilities for each token to be either a start of answer or an end of answer token. The answer is selected text span for which the combined start and end probabilities are the highest. Thresholding this probability allows to be stricter or more lenient on the quality of the answer, and can be optimized. The optimum F1 score is around a threshold of 0.3.



The best accuracy is chosen around 0.3. Note however that the F1 score stays constant between 0 and 0.3. This is a sign that the output probabilities of the model are not a good metric to discriminate good from bad answers. The most likely reason for this is that the generalization from SQuAD like question-answer pairs to CO2 questions creates confusion in the metric.

4.3.6 What is the best top-k retrievers ?

This experiment looks at the impact of the number of candidates that are taken from the shallow retriever that uses TF-IDF. So for this experiment, we switch to full-sized PDF documents.



As expected, the higher the top-k-retriever value, the better the precision and recall. However it becomes computationally too expensive to run experiments with more than 20 top-k-retrievers.

4.3.7 What is the best Table-To-Text translation ?

	Unnamed: 0	2018 Alphabet Carbon Footprint (MT CO2e)	Unnamed: 1
0	Scope 1	Location-based	Market-based
1	NaN	63,521	63,521
3	Scope 2	Location-based	Market-based
4	NaN	4,344,686	684,236
6	Subtotal Scope 1 + 2	4,408,207	747,757
8	Scope 3	Location-based	Market-based
9	NaN	150,467	150,467
11	NaN	TOTAL FOOTPRINT 4,558,674	898,224
13	Biogenic Emissions	Location-based	Market-based
14	NaN	22,862	22,862

Figure 4.14: Example of a table extracted from the Amazon sustainability report

Translation (single page PDF)	found one correct	precision	recall	F1
erasing the table	0.05	0.05	0.0125	0.02
row has value _ for column	0.7	0.55	0.13	0.21
column has value _ for row	0.75	0.58	0.11	0.18
row and column have value _	0.7	0.58	0.12	0.19
row has value _	0.75	0.56	0.14	0.21
column has value _	0.65	0.55	0.11	0.18

Table 4.2: The experiments with different table translations. “row” is replaced with the name of the row and “column” is replaced with the name of the column. _ is replaced with the cell value.

Erasing the tables from the input to the model makes the accuracy drop a lot, unsurprisingly. The chosen translation has a low impact on the accuracy.

4.3.8 What is the best question ?

Question (single page PDF)	foc	precision	recall	F1
What are the total CO2 emissions?	0.7	0.55	0.11	0.18
What are the total CO2 emissions in metric tons?	0.75	0.56	0.12	0.20
What is your footprint?	0.65	0.56	0.09	0.17
What is your footprint in metric tons?	0.75	0.55	0.12	0.20
What are the ghg emissions?	0.7	0.6	0.11	0.18
What are the ghg emissions in metric tons?	0.8	0.63	0.14	0.23
How much greenhouse gasses does your company put in air?	0.75	0.6	0.12	0.20
What is your footprint or ghg emissions in metric tons?	0.8	0.61	0.13	0.22
What is your footprint or ghg or co2 emissions in metric tons?	0.29	0.05	0.06	0.06
footprint ghg co2 emissions?	0.55	0.44	0.06	0.11

Notice that adding more words to the query does not necessarily result in a larger F1 score. The before last question for example gets a smaller F1 score, even though it contains more keywords.

4.3.9 Test set Accuracy

type	found one correct	precision	recall	F1
Single page PDF	0.7	0.5	0.13	0.20
Multiple pages PDF	0.4	0.25	0.07	0.11

4.3.10 Discussion

Using the PDF file as input to a BERT text based question answering system yields extremely low accuracies. This can be improved by using document layout analysis and table extraction. But the overall accuracy remains low, even when using the "perfect" tables from the dataset. This seems surprising as one could expect that the language model would adapt better and understand the sentences that were built from the tables, and requires further investigation.

Training procedure is not adapted to tabular data

With the very low accuracies in mind, we conduct the following experiment. A SQuAD-like dataset is generated with the tables in the CO2 dataset and by translating them to text. The aim of the experiment is to overfit this small dataset. In the first setting, we allow for multiple correct answers per $< question, passage >$ tuple. This happens often when extracting CO2 data as the numbers are all aggregated in the same table. In the second setting we only allow for one answer per $< question, passsage >$ tuple in the training data, and remove the other answers.

The result of the experiment is that the training error drops to almost zero after one epoch for the tuples $< question, passage >$ with a single answer, but stays constant for the tuples with multiple answers. This can be explained by the current training setup, which expects one correct answer per training paragraph. When the same training example is showed multiple times, but with a different answer, it causes the model to oscillate between the different answers.

This also means that the BERT model has been trained under a setup where there is only one correct answer per passage, and thus has learned to output only one correct answer per input. This works well on textual data as it is very unlikely to encounter two different and plausible answers in a short text span. But for tabular data, it happens very often, especially with the CO2 data. This could also explain the relatively high "Found One Correct" score combined to a very low F1 score of the system, meaning the system can find one correct answer but not multiple.

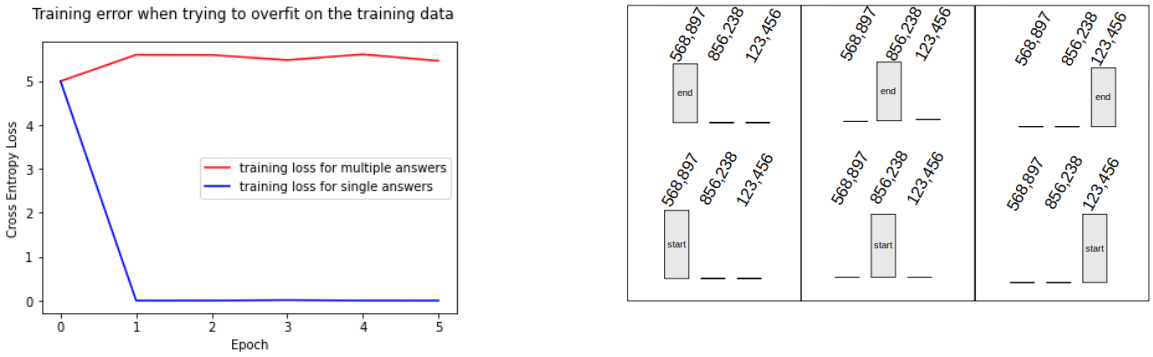
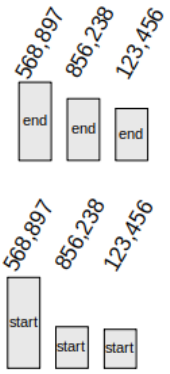


Figure 4.15: A SQuAD-like dataset was generated from the CO2-dataset. The training loss drops for training examples with single answers, but not when multiple answers are used. The image on the right shows the reference output probabilities used to compute the loss on successive training samples with the same paragraph but a different answer, which causes the model to oscillate between different answers.

Decoding procedure is not adapted to tabular data

When decoding the output probabilities of BERT, the question answering system uses a greedy approach that maximizes the combined start- and end-of-answer probability. This can cause multiple answers to be ouputted at once, and everything in between them too.



Rank	Answer
1	568,897
2	568,987 856,238
3	568,987 856,238 123,456

Figure 4.16: Greedy decoding of the output probabilities with multiple contiguous answers causes wrong answers when the start index and end index of different answers are selected.

The tokenization scheme is not adapted to numerical values

The numbers are split around commas and points, which does not make sense. It would be smarter to have one token per number. In practice, by observing the answers given by the system, one can see that this confuses the model as it will sometimes output only part of a number, which will be counted as wrong. For example, “635” instead of “635,587”.

Initial text	Tokenized (BPE)
Scope	Scope
1	Ġ1
has	Ġhas
value	Ġvalue
1,019,696	[Ġ1,',', 019, ',', 696]
for	Ġfor
CO2	[ĠCO, 2]
mtons	[Ġm,tons]

Figure 4.17: Tokenization of the sentence "Scope 1 has value 1,019,969 for CO2 mtons". The tokenizer uses special character Ġ for spaces and Ċ for new lines. This shows that numbers don't get tokenized properly.

4.3.11 Improvements

Improving the system could be done by training it on a large CO2 dataset, adapting tokenization for numbers, changing the training procedure and changing the decoding scheme. The details of these improvements are left for the corresponding section in LayoutLM. This is because, given additional training data, it seems more interesting to go with a one-stage system than with the pipelined architecture that was experimented in this section. Adding document layout analysis, table extraction and question answering in sequence makes it hard to train the system.

4.4 LayoutLM

We use a LayoutLM implementation trained on the DocVQA dataset without the image embeddings. The implementation is adapted to output the 10 best candidates for each question-document pair. Also, a filtering step to extract only digits is applied on the output of the question answering system.

4.4.1 Test set accuracy

PDF	found one correct	precision	recall	F1
Single page	0.5	0.43	0.11	0.18
Multiple pages	0.5	0.35	0.11	0.16

4.4.2 Discussion

The model suffers from the same problems as the RoBERTa question answering system with DLA preprocessing, i.e. a low recall and average precision. Since it is also a BERT-based model, the same conclusions regarding tokenization, training and decoding procedure can be drawn.

Answer for Amazon PDF by LayoutLM	Correct answer	Problem
51.17	51.17	correct
51	51.17	tokenization
. 17	51.17	tokenization
51.17 our carbon ... is equal to 122.8	51.17 or 122.8	decoding
8.01 hardware,equipment ... footprint 51.17	8.01 or 81.17	decoding
4,558,674 89	4,558,674 or 898,224	decoding & token

Table 4.3: The following table shows part of the response of LayoutLM to the questions "total footprint?" for the report of Amazon. Tokenization splits numbers on commas and points, which causes the model to return partial answers. Another error is that the model selects the start-of-index of a digit and the end-of-index of another digit, causing the text in between the two digits to be selected as part of the answer as well.

4.4.3 Improvements

Dataset

The most obvious way to improve would be to train it on a large CO2 dataset. This was not possible because of the current format of the CO2 dataset. In order to finetune LayoutLM on the CO2 dataset, the annotations of the dataset have to contain the bounding boxes of the CO2 emissions in the PDF file as well as their index in the sequence of tokens. In order to build such a dataset, one would need an efficient document annotator that supports bounding box annotations on PDF documents. Adding image embeddings could also increase accuracy.

Tokenization

In order to solve the tokenization problem, one could apply a preprocessing step before the tokenizer that replaces all numerical values with the token $< NUMBER >$. This would enable the model to learn a shared representation between all numbers. It could however remove some information, because the model would not be able to distinguish 2019 (which is a year) from 165,589 which is a number. So a rule based tokenization scheme could be applied that keeps years as they are.

Decoding scheme

When tables are queried in a document, it can happen that multiple answers are correct, especially in the case of CO2 emissions. Using a start-of-answer and end-of-answer decoding principle can cause mistakes by swapping start and end indices between different answers. Therefore, an alternate decoding principle could be used where the output probabilities correspond to "part of the answer" or not. The answer could then be extracted by thresholding the output probabilites and selecting the answer with the highest probability.

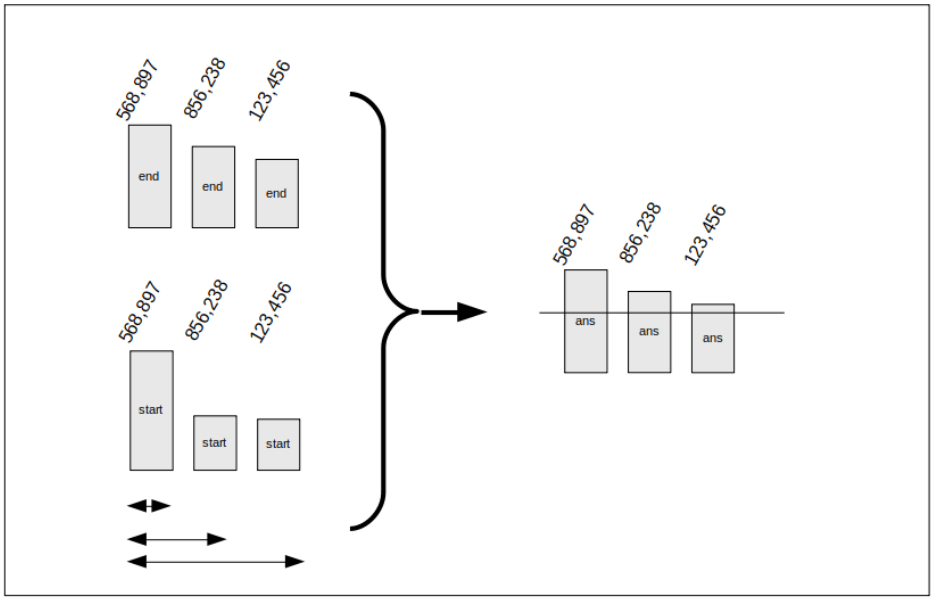


Figure 4.18: This shows the old decoding scheme based on start and end of answer probability distributions. The grey boxes represent probabilities. To the right is the proposed alternative scheme that would be more appropriate for numbers. Knowing that answers can only be a single token, and not a text-span such as in textual question answering systems, the answer could be decoded as a threshold to a single "Answer" probability distribution.

Training procedure

The training procedure would have to be adapted to the decoding scheme. This means training all correct answers at once together with a multi-label loss function.

4.4.4 Annotation tool

One of the big difficulties of this work is not having a large labelled dataset in the correct format. This was due to the fact that the data needed to be collected first, and then annotated manually. The annotations scheme was not compatible with the index-based training methods of language models.

However, research in document understanding has led to annotation tools that could be used for further work. The Allen Institute for Artificial Intelligence released PAWLS, an open source annotation tool for efficiently annotating PDF files and relationships withing the document (Neumann, Shen, and Skjonsberg 2021). Other commercial solutions exist such as IBM Watson's smart document understanding feature and TagTog released a Beta PDF annotation tool. Such tools would allow to annotate a decent sized dataset for CO2 extraction.

Chapter 5

Conclusion

In the context of ESG data extraction, this work presents methods to extract CO2 emissions for PDF documents. We first demonstrate a simple algorithm that partitions the PDF into buckets in order to extract numerical values. This is compared to a textual question answering system modified for tabular data and a layout-aware question answering system called LayoutLM. The GRID method performs better than the question answering systems. The weaknesses of the question answering system in the context of numerical tabular data are identified and improvements are proposed. Finally, let us answer the questions that were asked in the introduction.

What is the best method for extracting CO2 data, without large training set, from PDF business documents?

From the methods that were tried in this study, and under the constraint of not having any training data, the best method to extract CO2 data from business documents is the method based on table extraction called GRID.

How could the selected method be improved?

The GRID method has a lot of room for improvement such as choosing the partitioning of the PDF in a smarter way, developing more accurate distance metrics and filtering rules.

What method looks the most promising in the close future?

The convenience of providing textual queries, as well as the good performance on a dataset such as DocVQA make layout-aware language models appealing. Given enough training data and the adaptation of the training and tokenization to numerical data, they seem the most promising method.

Can traditional language models be adapted to tabular data, and in doing so, can we leverage all the existing open source libraries ?

Although zero-shot capabilities of recent question answering systems are impressive, they don't generalize well when provided textual representations of tables. Language models suffer in particular of a very low recall. This can partly be explained by the tokenization that is not adapted to numerical data and the decoding scheme that is not adapted to detecting many answers in one same paragraph.

Bibliography

anisha2102/docvqa: Document Visual Question Answering (May 31, 2021). URL: https://github.com/anisha2102/docvqa (visited on 05/31/2021).

Attention? Attention! (June 24, 2018). URL: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html (visited on 05/31/2021).

Camelot: PDF Table Extraction for Humans — Camelot 0.8.2 documentation (Jan. 29, 2021). URL: https://camelot-py.readthedocs.io/en/master/ (visited on 05/30/2021).

Denk, Timo I. and Christian Reisswig (2019). *BERTgrid: Contextualized Embedding for 2D Document Representation and Understanding*. arXiv: 1909.04948 [cs.CL].

Devlin, Jacob et al. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL].

Docparser - Document Parser Software (May 28, 2021). URL: https://docparser.com/ (visited on 05/30/2021).

Document AI Solution Google Cloud (May 10, 2021). URL: https://cloud.google.com/document-ai (visited on 05/30/2021).

Document Intelligence Workshop at NeuriPS 2019 (May 29, 2021). URL: https://sites.google.com/view/di2019 (visited on 05/29/2021).

Documents and Block Objects - Amazon Textract (May 29, 2021). URL: https://docs.aws.amazon.com/textract/latest/dg/how-it-works-document-layout.html (visited on 05/30/2021).

Elomaa, T. (2013). "Algorithmic extraction of data in tables in PDF documents." In:

facebookresearch/detectron2 (June 12, 2021). URL: https://github.com/facebookresearch/detectron2 (visited on 06/12/2021).

Form Recognizer Microsoft Azure (May 30, 2021). URL: https://azure.microsoft.com/fr-fr/services/form-recognizer/#features (visited on 05/30/2021).

Garncarek, Łukasz et al. (2021). *LAMBERT: Layout-Aware (Language) Modeling for information extraction*. arXiv: 2002.08087 [cs.CL].

Haystack - Question Answering (May 28, 2021). URL: https://haystack.deepset.ai/ (visited on 05/31/2021).

Honda, Hiroto (July 7, 2020). *Digging into Detectron 2 — part 1 — by Hiroto Honda — Medium*. URL: https://medium.com/@hirotoschwert/digging-into-detectron-2-47b2e794fabd (visited on 05/30/2021).

hpanwar08/detectron2: Detectron2 for Document Layout Analysis (June 12, 2021). URL: https://github.com/hpanwar08/detectron2 (visited on 06/12/2021).

Karpukhin, Vladimir et al. (2020). *Dense Passage Retrieval for Open-Domain Question Answering*. arXiv: 2004.04906 [cs.CL].

Krotov, Vlad and Leiser Silva (Sept. 2018). "Legality and Ethics of Web Scraping." In:

Liu, Yinhan et al. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv: 1907.11692 [cs.CL].

Neumann, Mark, Zejiang Shen, and Sam Skjonsberg (2021). *PAWLS: PDF Annotation With Labels and Structure*. arXiv: 2101.10281 [cs.CL].

Paliwal, Shubham et al. (2020). *TableNet: Deep Learning model for end-to-end Table detection and Tabular data extraction from Scanned Document Images*. arXiv: 2001.01469 [cs.CV].

pdfminer · PyPI (Nov. 25, 2019). URL: https://pypi.org/project/pdfminer/ (visited on 05/31/2021).

Powalski, Rafał et al. (2021). *Going Full-TILT Boogie on Document Understanding with Text-Image-Layout Transformer*. arXiv: 2102.09550 [cs.CL].

Question Answering with a Fine-Tuned BERT · Chris McCormick (Mar. 10, 2020). URL: https://mccormickml.com/2020/03/10/question-answering-with-a-fine-tuned-BERT/ (visited on 06/05/2021).

Rajpurkar, Pranav, Robin Jia, and Percy Liang (2018). *Know What You Don't Know: Unanswerable Questions for SQuAD*. arXiv: 1806.03822 [cs.CL].

Rossum's line item extraction from invoices (Aug. 8, 2018). URL: https://rossum.ai/blog/update-line-item-extraction-invoices/ (visited on 06/09/2021).

Schreiber, Sebastian et al. (2017). "DeepDeSRT: Deep Learning for Detection and Structure Recognition of Tables in Document Images." In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 01, pp. 1162–1167. DOI: 10.1109/ICDAR.2017.192.

Schuster, Mike and Kaisuke Nakajima (2012). "Japanese and Korean voice search." In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). *Neural Machine Translation of Rare Words with Subword Units*. arXiv: 1508.07909 [cs.CL].

Synced (Sept. 10, 2017). *Language Model: A Survey of the State-of-the-Art Technology — by Synced — SyncedReview — Medium*. URL: https://medium.com/syncedreview/language-model-a-survey-of-the-state-of-the-art-technology-64d1a2e5a466 (visited on 05/29/2021).

Tabula: Extract Tables from PDFs (June 16, 2013). URL: https://tabula.technology/ (visited on 05/30/2021).

Transformers Huggingface (May 28, 2021). URL: https://huggingface.co/transformers/ (visited on 05/30/2021).

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].

Want to use our data? – Common Crawl (May 29, 2021). URL: https://commoncrawl.org/the-data/ (visited on 05/29/2021).

Watson Discovery IBM (May 17, 2021). URL: https://www.ibm.com/cloud/watson-discovery (visited on 06/11/2021).

What Is Text Mining, Marti Hearst (Nov. 11, 2005). URL: https://www.jaist.ac.jp/~bao/MOT-Ishikawa/FurtherReadingNo1.pdf (visited on 05/31/2021).

Xu, Yiheng et al. (2019). "LayoutLM: Pre-training of Text and Layout for Document Image Understanding." In: *CoRR* abs/1912.13318. arXiv: 1912.13318. URL: http://arxiv.org/abs/1912.13318.

Zhong, Xu, Jianbin Tang, and Antonio Jimeno Yepes (2019). "PubLayNet: Largest Dataset Ever for Document Layout Analysis." In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1015–1022. DOI: 10.1109/ICDAR.2019.00166.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**