Publication date predictor

NLP in Industry - final project

Hanna Brinkmann, Zofia Milczarek, Alexandre Nechab, Joanna Radoła

M2 Linguistique Informatique 2024/2025

Contents

1 Introduction

Dematerialized documents are, for some companies, an important source of data to predict future activity or development. In order to obtain a prediction that is as reliable as possible, it is paramount for those companies to know the date of the document. Not every document that has recently appeared on a website is brand new. Depending on several factors, it is possible that it is several years old. At this point, the present project comes into play. We propose a method to automatically determine the publication date of the document. This way, the companies save time that would have previously been spent on annotating the documents.

In our approach to the task, we use a Large Language Model (Llama) to pass it a prompt containing the most relevant chunks of the document and to get the prediction, which we then postprocess to get the desired format and compare it to the gold standard. We also tested a visual language model. Our best scores are a result of fine-tuning LLaMA 3.1 8B with QLoRA. The GitHub repository for this project is available here. The dataset, labelled together with other students, is available on HuggingFace under this link. It has 3 columns Text, Gold published date and url; and contains 500 rows.

2 Methodology

2.1 Data preprocessing

We tested two different types of preprocessing before performing the predictions:

1. extracting sentences containing dates from documents with regex,
2. taking the beginning and end of documents.

2.1.1 Regex chunks

First, we used a regular expression to extract the chunks surrounding dates from the .txt versions of the files. We decided to extract the 50 characters preceding and following the date. The regex was the following:

```
"\b(\d{1,2})\s*[./-]?\s*(?:([a-zéû]{3,10})|(\d{1,2}))\s*[./-]?\s*(\d{4})?\b"
```

Regex breakdown:
1. 1 or 2 digits, followed by optional space(s), separated by '/', '.' or '-', again followed by optional space(s). (to express the DD/ part)
2. Either the sequence from 1. again, or an alphabetical string between 3 and 10 characters (month name in French, including diacritics present in month names, i.e. 'é' and 'û'), followed and preceded by optional spaces. (to express MM/)
3. Optional 2 or 4 digits for the YYYY part.
The LLM received the complete dataset as we only used regex chunks for inference and not for fine-tuning a model.

2.1.2 Head-tail chunks

In this approach, we extracted the first 3,000 characters from both the beginning and the end of each .txt version of the documents. This method was motivated by the observation that the publication dates were almost always located there. Compared with regular expressions, this approach captures a better context to help the model extract the correct publication date. We should note, however, that text chunks have more tokens. Therefore, more computation time is needed to run the predictions.

2.2 Prompt Engineering

2.2.1 LLMs

We used two different methods to prompt the LLM. First, we tried the Llama 3.2 model with checkpoint `unsloth/Llama-3.2-3B` and 8bit quantization. Second, we tried `meta-llama/Llama-3.1-8B` model with the Ollama class from LlamaIndex.

In order to generate a prompt template, we used Jinja2's Template method. Below are the different templates we used:

```
    # Ollama prompt
    prompt_template = Template(
    """
    <|begin_of_text|>
    <|start_header_id|>user<|end_header_id|>
    Document : {{document}}
    What is the publication date? Output as a structured JSON
    object with format DD/MM/YYYY.
    <|eot_id|>
    {'publication_date':
    """
 )

    # HuggingFace prompt
    prompt_template = Template(
    """
    You are an expert in structured data extraction.
    The document is: "{{ document }}"

    Extract and output only the **publication date** of the document
    in the format DD/MM/YYYY. Do not include any additional text or
    context, just the date. The answer has to be 10 characters long.
    Publication date:
    """
)
```

Jinja2 allows us to pass variables to the prompt without having to change it every time. That way we can pass the previously generated strings directly and call `prompt_template.render(**input)` to generate the prompt. Our prompt uses meta-tokens recommended for Llama and it is clearly structured in order to get the expected result.

The next step was to predict the publication date for each file in the dataset. We gave the prompt the regex chunks prepared above as {document} variable. As we stored the dataset in a pandas dataframe, we saved the prediction in another column on the same line. The functions we used for this are:

```
def predict_date(document, temperature=0.5):
    input = {'document': document}
    prompt = prompt_template.render(**input)
    output = llm_complete(prompt, max_tokens = 10, temperature=temperature)
    date = re.findall(r"\d{2}/\d{2}/\d{4}", str(output))
    if date: return date[0]
    else: return str(output)

test['prediction'] = test.apply(lambda x: predict_date(x['regex_chunks']), axis=1)
```

Sometimes, the date format was not as expected. For those cases, we post-processed the dates, converting them to the expected date format of DD/M-M/YYYY.

2.2.2 VLM

Since some documents in the dataset contain a date in the pdf, but not in the text form, we decided to experiment with a visual language model. Due to computational limits, we tried a small, 2B paramater model from the `HuggingFaceTB/SmolVLM-Instruct` checkpoint.

As input to the model, we converted the 1st page of each pdf file into an image, and then used that as the context to the model. The methods in `preprocessing_VLM.py` download the PDF files and with the use of the `fitz` library we turn the 1st page of the PDF into an image. This image is then loaded into the model as part of its context.

Some of the documents were not possible to fetch using this method. We ignored these rows during evaluation, leaving us with 462 rows.

The general prompt looks like this:

```
messages = [{
                "role": "user",
                "content": [
                    {"type": "image"},
                    {"type": "text", "text": "What is the publication
                    date of this document? Answer in a numerical
                    date format YYYY-MM-DD."}
```

```
                        ]
                    }
                ,]
)
```

Similarly to text-based LLMs, the model struggled with outputting a correct date format, often outputting the date as DD/MM/YY, DD-MM-YYYY, or in a written manner, as '10 Janvier 2022'. We used the `dateparser` library to postprocess these irregular date formats. This covered many irregular cases, but sometimes the model's outputs were not valid dates in any way and were impossible to be parsed.

The model performed very poorly, achieving only 36.7% accuracy on exact matches.

2.3 Finetuning Llama 3.1 8B

We used the `unsloth` and `trl` libraries to fine-tune a Llama 3 8B model in 4bit quantization, using QLoRA. We used the model from the `unsloth/llama-3-8b-bnb-4bit` checkpoint.

2.3.1 Data Preprocessing

Since in the Prompt Engineering section we found that using a Llama 8b model performed the best, and there was little to no difference between using Head and Tail chunks or Regex chunks for the context (see Section 2.1.2). We took the text available to us under the links in the 'text version' column of the supplied dataset. Unfortunately, some links were not possible to fetch, which left us with a dataset of 469 rows. We use 328 rows (70%) for fine-tuning and 141 rows (30%) for evaluation.

Each example was transformed into the following prompt for the finetuning:

```
[{'role': 'user', 'content': f'{context}\nWhat is the
publication date of the document? Output as a structured
JSON object with a format DD/MM/YYYY.'},
{'role': 'assistant', 'content':f'{gold_date}'}]
```

Where the context is the concatenated first and last 3000 characters of each document. This prompt is then converted into one string that includes all the special tokens and applies the necessary formatting using the `apply_chat_template` method. Thus preprocessed dataset is now ready for training.

2.3.2 Fine-tuning using QLoRA

The fine-tuning was done on 3 epochs with learning rate = 2e-4, warmup steps = 5, weight decay = 0.05 and using the AdamW8bit optimizer. For QLoRA hyperparameters, we chose rank = 16 and alpha = 16. We finetune on responses only, meaning that the backward pass is done only for the part of the prompt with the role 'assistant'. The resulting adapters were put on the HuggingFace hub and can be accessed through the `zmilczarek/llama3_8b-finetuned -nlp_industry-adapters` checkpoint. The adapters can be loaded in the followin 2 ways:

- `unsloth` library

```
from unsloth import FastLanguageModel, get_chat_template

model, tokenizer = FastLanguageModel.from_pretrained(
        model_name = "zmilczarek/llama3_8b-finetuned-nlp_industry-adapters",
        max_seq_length = 2048,
        dtype = None,
        load_in_4bit = True,
    )
```

- `PEFT` and `Transformers` libraries

```
from peft import AutoPeftModelForCausalLM
from transformers import AutoTokenizer
model = AutoPeftModelForCausalLM.from_pretrained(
    "zmilczarek/llama3_8b-finetuned-nlp_industry-adapters",
    load_in_4bit = True,
)
tokenizer = AutoTokenizer.from_pretrained(
        "zmilczarek/llama3_8b-finetuned-nlp_industry-adapters")
```

2.3.3 Inference and post-processing

For inference, each row was turned into the following prompt:

```
[{'role': 'user', 'content': f'{context}\nWhat is the
publication date of the document? Output as a structured
JSON object with a format DD/MM/YYYY.'}]
```

The model output was also restricted to generate at most 13 tokens, since this is the number of tokens of the ideal answer, i.e. {`'predicted_date' : '23/02/2023'`}. Sometimes the answer is {`'predicted_date' : '2023'`} `<some_token> <some_token>....` In that case, we ignore the output after the closing bracket }.

In general, this approach lead to very consistent outputs of the model and there was no need for more complex post-processing.

On the test set, the model achieved 71% on exact-match accuracy (day-month-year precision). The score was 80% for accuracy with month-year precision.

2.4 Evaluation and Results

Our basic evaluation measure is accuracy. We compare the annotated gold dates with the predicted dates and calculate the proportion of exact matches.

For Llama models, we report the results on 2 approaches: Regex and Head-Tail. For Llama3-8B finetuned, we only report Head-Tail since the model was fine-tuned on this approach. Additionally, the results of this model are computed based on 141 rows of the test set.

For the VLM, we report the scores based on the 1st page.

2.4.1 Alternative evaluation metrics considered

The accuracy of our best model on the task of predicting the correct year of publication was 92.8%, and for the correct month and year it was 82.0%.

As an alternative to simple accuracy, we also computed the mean number of days between the prediction and the reference. We obtained a mean variance of 84 days. However, this is not a very relevant result, as some outliers (one or two predictions wrong by more than a year) caused the mean to rise significantly.

For a relaxed condition of what consists in a correct prediction, we also compute the accuracy of our best model counting predictions that are less than 5 days apart from the reference as correct. The relaxed accuracy therefore obtained was 76.3%.

All these results are in Table 1.

Model	**Strict Accuracy**		**Relaxed Accuracy**		**MM/YY Accuracy**	
	Regex	Head-Tail	Regex	Head-Tail	Regex	Head-Tail
Llama3-3B	0.47	0.53	0.51	0.57	0.63	0.63
Llama3-8B (base)	0.63	0.63	0.67	0.67	0.74	0.73
Llama3-8B (fine-tuned)		**0.71**		**0.76**		**0.82**
SmolVLM-1B	0.36		0.44		0.54	

Table 1: Accuracy results across strict accuracy (exact date match), relaxed accuracy (prediction within 5 days of the reference date) and MM/YY accuracy. We also include the results of the VLM ran on the first page of the pdfs.

3 How to run - CLI Inference

We added a CLI tool which allows to predict the dates over the whole dataset.

The `finetuned_llama_inference.py` and `preprocessing_finetune.py` files can be either used as a python module or it can be used as a command-line script. `preprocessing_finetune.py` contains methods for obtaining the dataset with a column 'text' that contains the start and end chunks of the document. After the dataset is processed into the appropriate format, `finetuned_llama_inference.py` can be used to automatically label a whole dataset.

Steps to follow after cloning the repository:

1. First, we need to install the necessary libraries:

   ```
   $ pip install -r requirements.txt
   ```

2. To get the dataset in the appropriate format, the following CLI command can be used:

   ```
   $ python preprocessing_finetune.py \
      --input-path original_dataset.csv \
      --output-path df_for_finetuning \
      --out-type csv \
      --length 3000 \
   ```

   Details about the parameters:

   Only the `--input-path` and `--output-path` flags are necessary. The file specified by the `--input-path` has to contain the columns 'url' (original url of the document) and 'text version' (datapolitics link to a text version of the document). It can be in .pkl or .csv format. By default `--out-type` is set to csv, but can also be pkl (pickle) or dataset (as in Huggingface Datasets object). `--length` is 3000 by defualt and it defines the length of start and end chunks that are taken from each document. If the script is ran on the whole dataset with the goal of genrating a train-test split, it has to be run without the `--no-split` flag. If the goal is to only use inference on the dataset, the script should be run with the `--no-split` flag.

3. After `preprocessing_finetune.py` has been run and the data has been processed, the command below can be used to predict the dates for the whole dataset. `finetuned_llama_inference.py` will take in the dataset with the documents in text format, label it using the model from the checkpoint and save the labels to a new csv or pkl file.

   ```
   $ python finetuned_llama_inference.py \
     --input-path df_for_finetuning/df_test.csv \
     --output-path df_labeled.csv \
     --model-checkpoint zmilczarek/llama3_8b-finetuned-nlp_industry-adapters \
     --gold-labels 'Gold published date'
   ```

   Details about the parameters:

   The `--model-checkpoint` and `--gold-labels` flags are optional. If no model checkpoint is used, `zmilczarek/llama3_8b-finetuned-nlp_industry-adapters` is the default. If the `--gold-labels` parameter is passed, the script will report 3 accuracy scores : exact matches, month-year matches and year matches. If not, the script will only save the predicted dates and it will not compute any metrics.

4. *(optional)* After getting new inferences and returning the output in the .pkl format, it can be evaluated by setting `path_to_predictions_df` (at the beginning of the `eval.ipynb` notebook) to be the path to the output file.

4 Conclusion

Despite the fact that the task turned out to be quite complex, we were able to obtain around 80% accuracy with the fine-tuned LLaMA 8B. All in all, our best result was 71% strict accuracy, 0.76 relaxed accuracy and 82% MM/YY accuracy.

The biggest challenges and limitations we faced was the limited context length in LLMs, lack of computational resources to test more options and bigger models, as well as the small size of the dataset. Prompt Engineering, while promising, wasn't enough to get the smaller size model to achieve high accuracy. This method also came with the problem of models' inability to conform to the expected format. The QLoRA fine-tuning approach improved model outputs on both accounts : the accuracy jumped to 71%, and the model learned to give the answer in the desired format.