Analysis of PDF CVs using RShiny and LM Studio

Jesús Veiga Morandeira

2026-01-10

Introduction

This document provides an in-depth overview of a Shiny application built to analyze PDF-based CVs, focusing on extracting key details like name, contact information, email, education, and work experience. The application leverages various R libraries and an external language model (LM) for structured analysis.

The main objectives of this document are to:

- Explain the technical setup for PDF extraction.
- Detail the usage of an OCR system for text recognition.
- Describe the interaction with a language model API for CV parsing.

State of the Art

Analyzing PDF-based resumes is challenging due to the unstructured nature of the content. Several solutions involve Optical Character Recognition (OCR) systems and Language Models (LMs) to improve accuracy:

- **OCR Tools**: `tesseract` is employed to convert non-text PDF content to digital text.
- **Language Models**: This solution integrates `Llama-3-Groq-8B` from `LMStudio-Community`, an advanced transformer-based LM suited for entity extraction from unstructured text.

Development

This section provides a line-by-line explanation of the code structure used in the application.

Libraries and API Setup

```
library(shiny)
library(tidyverse)
library(DT)
library(purrr)
library(httr)
library(jsonlite)
library(pdftools)
library(tesseract)
library(openxlsx)
```

These libraries enable core functionality:

- **shiny** for UI and server components.
- **tidyverse** and **purrr** for data manipulation.
- **DT** for table display.
- **httr** and **jsonlite** for API calls.
- **pdftools** and **tesseract** for PDF and OCR processing.

What is OCR?

Optical Character Recognition (OCR) is a technology that converts various types of documents, such as scanned paper documents, PDF files, or images taken with a digital camera, into editable and searchable data. This process allows text to be extracted from images, making it possible to automate data entry and processing tasks. It is important in our CV Analysis because sometimes we may receive CVs in PDF format that come from a JPG or PNG image. In that case, R would not be able to transform it into plain text (which is the text format needed for LLM) using only "pdftools," since this library is oriented to handle PDF documents containing text.

How Does OCR Work?

- **Image Preprocessing**: The first step involves preprocessing the image to improve its quality for OCR. This may include converting images to grayscale, applying filters, and adjusting contrast.
- **Text Detection**: The OCR software identifies regions in the image that contain text. This process may involve recognizing lines, words, and characters.

- **Character Recognition**: After detecting text regions, the OCR engine analyzes the shapes of characters within those regions and matches them against known character patterns.
- **Post-processing**: Finally, the recognized text may undergo post-processing to correct errors and improve formatting. This step can involve spell-checking and using dictionaries to refine the output.

API Configuration

```
base_url <- "http://localhost:1234/v1"
api_key <- "llama-3-groq-8b-tool-use"
myModel <- "lmstudio-community/Llama-3-Groq-8B-Tool-Use-GGUF"
```

Here, we configure the API endpoint and model, with Llama-3-Groq-8B-Tool-Use-GGUF specified for resume analysis. This setup helps ensure that data processing leverages the appropriate language model.

Functions for Text Extraction

1. Extracting Text with OCR

```
extract_text_with_ocr <- function(pdf_path, pages) {
  pdf_convert(pdf_path, format = "png", dpi = 300, pages = pages) %>%
    map(ocr, engine = tesseract_lang) %>%
    str_c(collapse = " ")
}
```

Explanation of Steps:

Function Definition:

The function `extract_text_with_ocr` takes two parameters:

- **pdf_path**: The file path of the PDF document to be processed.
- **pages**: A vector indicating which pages of the PDF to convert to images.

PDF Conversion:

The first line of the function calls `pdf_convert`, which is part of the `pdftools` library.
It converts the specified pages of the PDF into PNG images. The parameters specify that the output format should be "png" and the resolution should be set to 300 DPI (dots per inch) for better image clarity, which is crucial for effective OCR processing.

Text Extraction with OCR:

The resulting PNG images are then passed to the `map` function from the `purrr` package, which applies the `ocr` function to each image.
The `ocr` function utilizes the Tesseract OCR engine (initialized earlier in the code with `tesseract_lang`) to extract text from each image.

Concatenation of Text:

Finally, the extracted text from all pages is concatenated into a single string using `str_c(collapse = " ")`.
This results in a unified text output that represents the content of the specified pages of the PDF.

API Call Function

```
call_api <- function(full_text) {
  body <- list(
    model = myModel,
    messages = list(
      list(role = "system", content = "You are a CV analysis assistant..."),
      list(role = "user", content = full_text)
    )
  )
  response <- POST(url = paste0(base_url, "/chat/completions"),
                   add_headers(Authorization = paste("Bearer", api_key)),
                   body = toJSON(body, auto_unbox = TRUE),
                   encode = "json")
  return(content(response))
}
```

Explanation of Steps:

1. Function Definition
The function `call_api` is defined with a single parameter:

- **full_text**
  A string that contains the text extracted from the PDF, which is to be analyzed by the language model.

2. **Constructing the Request Body**:

- The body of the request is constructed as a list:
  - The model is set to the `myModel` variable, which specifies the language model to be used for analysis.
  - The `messages` list contains two entries:
    * The first entry has a role of "system", which is usually used to provide instructions or context to the language model. Here, it includes a prompt indicating that the model is a CV analysis assistant. In this case the prompt that appears in the code is an example code but obviously we can ask the LLM whatever we want.
    * The second entry has a role of "user", which contains the actual content of the CV (`full_text`) that needs to be analyzed.

3. **Making the API Call**:

- The `POST` function from the `httr` package is used to send the HTTP POST request to the specified API endpoint (base_url combined with `/chat/completions`).
- The request includes:
  - `add_headers`: This sets the authorization header using the Bearer token (`api_key`), which is necessary for secure access to the API.
  - `body`: The request body, which is converted to JSON format using `toJSON(body, auto_unbox = TRUE)` to match the API's expected input structure.
  - `encode = "json"`: This specifies that the body content is in JSON format.

4. **Returning the Response**:

- The function retrieves the content of the response from the API call using `content(response)`.
- This content typically contains the analysis results from the language model, which is then returned as the output of the `call_api` function.

Function to Process Each PDF CV

```
process_cv <- function(pdf_file) {
  pdf_text_pages <- pdf_text(pdf_file)
  empty_pages <- which(
```

```
    map_int(pdf_text_pages, nchar) == 0
  )
  text_with_text <- pdf_text_pages %>%
    keep(~ nchar(.) > 0) %>%
    str_c(collapse = " ")
  text_without_text <- if (length(empty_pages) > 0) {
    extract_text_with_ocr(pdf_file, empty_pages)
  } else { "" }
  final_text <- str_c(
    text_with_text, text_without_text, collapse = " "
  )
  message_content <- call_api(final_text)
  separated_content <- str_split(
    message_content, ";"
  )[[1]] %>% map_chr(trimws)

  if (length(separated_content) < 5) {
    separated_content <- c(
      separated_content, rep(NA, 5 - length(separated_content))
    )
  }

  data.frame(
    Name = separated_content[1],
    Contact_Info = separated_content[2],
    Email = separated_content[3],
    Education = separated_content[4],
    Experience = separated_content[5],
    stringsAsFactors = FALSE
  )
}
```

Explanation of Steps:

1. **Function Definition**:
   - The function `process_cv` is defined to process a PDF CV file and extract relevant details.
2. **PDF Text Extraction**:
   - The function starts by extracting text from the PDF using the `pdf_text(pdf_file)` function, which returns a list of strings where each string represents the text of a page.

3. **Identifying Empty Pages**:

   - The function identifies empty pages by using `which(map_int(pdf_text_pages, nchar) == 0)`. This line checks each page's character count and finds indices of pages that have no text.

4. **Text Concatenation**:

   - **Text with Content**: It filters out non-empty pages using `keep(~ nchar(.) > 0)` and concatenates their text into a single string called `text_with_text`.
   - **Text without Content**: If there are empty pages, it calls the `extract_text_with_ocr` function to retrieve text from those pages using OCR. If there are no empty pages, it sets `text_without_text` to an empty string.
   - **Final Text**: The complete text is then created by concatenating `text_with_text` and `text_without_text`, ensuring all available content is included.

5. **API Interaction**:

   - The concatenated text, stored in `final_text`, is passed to the `call_api` function. This function sends the text to the language model for analysis and retrieves the results.

6. **Parsing the Response**:

   - The API response, `message_content`, is split by semicolons (`";"`) into individual components, and any leading or trailing whitespace is removed using `map_chr(trimws)`.

7. **Handling Incomplete Data**:

   - The function checks the length of `separated_content`. If it has fewer than five elements, it appends `NA` values to ensure that the output always has five fields: Name, Contact Info, Email, Education, and Experience.

8. **Creating a Data Frame**:

   - Finally, the function returns a data frame containing the extracted information, with each piece of data assigned to the appropriate column.

User Interface

The user interface of the Shiny application is built using the `fluidPage` function, which creates a responsive layout that adjusts to various screen sizes. The UI consists of several key components:

UI Components

- **Styling**: Custom CSS is included in the `tags$head` section to enhance the aesthetics of the application. This includes styles for the body, title, sidebar, main content area, loader, and language switch flags.
- **Title Panel**: The `titlePanel` function displays the application title at the top of the page. It uses `textOutput` to allow dynamic updating of the title based on the selected language.
- **Sidebar Layout**: The layout is structured with a sidebar on the left and the main content on the right, created using `sidebarLayout`.
  - **Sidebar Panel**:
    - A file input (`fileInput`) is provided for users to upload PDF files.
    - An action button (`actionButton`) is available to trigger the CV processing.
    - A download button (`downloadButton`) allows users to download the processed data.
  - **Main Panel**:
    - This section contains a loading indicator (rendered using `uiOutput`) to inform users that the processing is ongoing.
    - A data table (`DTOutput`) displays the extracted information from the CVs.
- **Language Switch**: The language switch feature allows users to toggle between English and Spanish. The flags are interactive elements that use JavaScript to send input values back to the Shiny server for language updates.

Server Logic

The server function manages the application's logic and data processing:

Server Components

- **Reactive Values**:
  - `data`: Stores the extracted information from the CVs.
  - `lang`: Holds the current language setting.
  - `is_loading`: Indicates whether the application is currently processing data.
- **Text Definitions**: Two lists, `texts_en` and `texts_es`, store the text for different UI elements based on the selected language.
- **Update Function**: The `update_texts` function checks the selected language and updates the UI text accordingly.

- **Loading Indicator**: The `loader` UI element is displayed based on the `is_loading` reactive value, providing visual feedback during the CV processing.
- **Processing Action**: When the user clicks the process button, the application:
  - Validates that a file has been uploaded (`req(input$file)`).
  - Sets the loading indicator to TRUE (`is_loading(TRUE)`).
  - Uses `map_dfr` to apply the `process_cv` function to each uploaded file, combining the results into a single data frame (`final_data`).
  - Stores the processed data in the `data` reactive value and resets the loading indicator.
- **Data Table Rendering**: The processed data is rendered in a data table using `renderDT`, displaying the results with pagination.
- **Data Downloading**: The `downloadHandler` function enables users to download the extracted data as either a CSV or XLSX file based on their selection.

Running the Application

The application is launched with the `shinyApp(ui, server)` function call, which combines the user interface and server logic into a single Shiny application.

Tests

Test 1: Using Phi-3.1-mini-128k

The Phi-3-Mini-128K-Instruct is a 3.8 billion-parameter, lightweight, state-of-the-art open model trained using the Phi-3 datasets. This dataset includes both synthetic data and filtered publicly available website data, with an emphasis on high-quality and reasoning-dense properties. The model belongs to the Phi-3 family with the Mini version in two variants 4K and 128K which is the context length (in tokens) that it can support.

It is a reasonably compressed model for use in standard computers that do not require a very powerful GPU or graphics card. Nevertheless, the results it provides for our purposes are more or less acceptable. It makes some mistakes and leaves some fields blank that could be filled in by hand in the downloadable file.

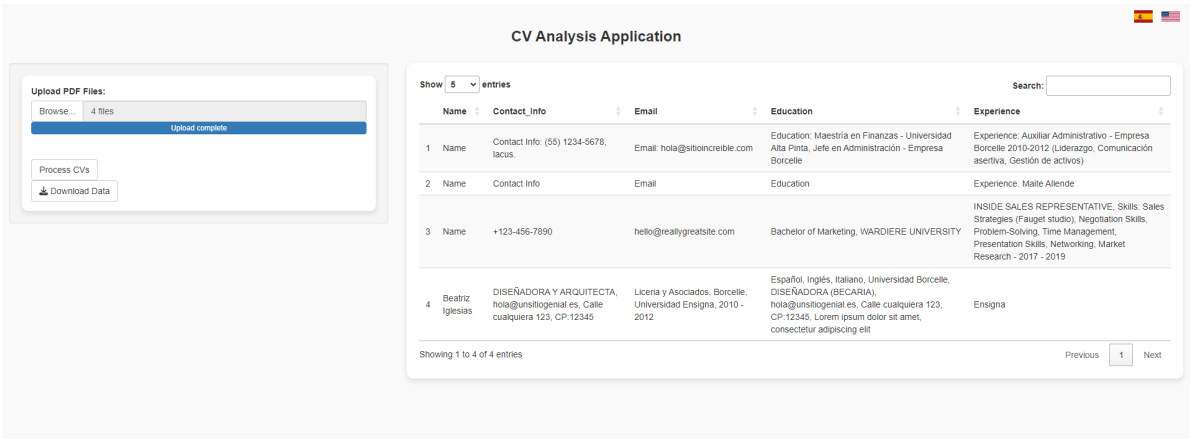

Figure 1: App Screenshot Results using Phi

As we can see, the results are not bad, although there are some errors, for example, the experience of the fourth curriculum is not correct.

Test 2: Using Llama-3-Groq-8B-Tool-Use-GGUF

This is a model designed for research and development in tool use and function calling scenarios. It excels at tasks involving API interactions, structured data manipulation, and complex tool use.

Note the model is quite sensitive to the `temperature` and `top_p` sampling configuration. Start at `temperature=0.5, top_p=0.65` and move up or down as needed. In this specific case, the temperature was fixed at 0.5.

It is a more powerful model than the previous one, although it is also heavier for the computer, so we must consider in each case and depending on the capacity we have, what is best for our use case.

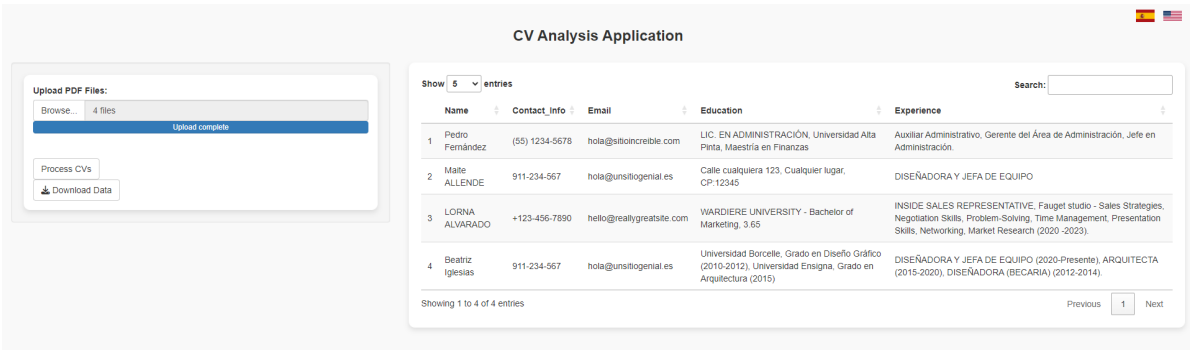

Figure 2: App Screenshot Results using Llama

As we can see the results are significantly better than with the previous model applied on the same test CVs. It still makes small errors but gives much clearer information.

Conclusion

This Shiny application serves as a powerful tool for analyzing PDF CVs, automating the extraction of key information with the help of OCR and a language model. Its intuitive interface and responsive design make it user-friendly, while the underlying code structure allows for scalability and further enhancements.

The application is useful for the automation of internal tasks such as the organisation of CV information as is the case. The good thing is that it is generalisable, we can include as many documents as we like and as many as we want, adapting the prompt to our needs and avoiding asking for very detailed and complex information, it will automate tasks satisfactorily, thus saving a lot of time.

Next Steps

Future developments could include:

- Integrating additional data validation checks to improve the robustness of the extracted information.
- Implementing user authentication to secure the application.
- Enhancing the data visualization capabilities to provide insights into the analyzed CVs or our preferred documents.

- Use of more powerful models such as the popular gpt 3.5 or gpt 4 although they are already pay-per-token models. It all depends on our purpose, if it is just to make simple prompts for automation tasks, we can continue with this type of models. However, it would be nice to try others and assign more complex tasks.

Bibliography

- **R Core Team. (2022).** *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. URL: https://www.R-project.org/
- **Chang, W., Cheng, J., Allaire, J., Xie, Y., & McPherson, J. (2022).** *Shiny: Web Application Framework for R.* R package version 1.7.2. URL: https://CRAN.R-project.org/package=shiny
- **Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019).** *Welcome to the Tidyverse.* Journal of Open Source Software, 4(43), 1686. DOI: 10.21105/joss.01686
- **Hocking, W. (2021).** *DataTables.* URL: https://CRAN.R-project.org/package=DT
- **Wickham, H., François, R., Henry, L., & Müller, K. (2022).** *Dplyr: A Grammar of Data Manipulation.* R package version 1.0.10. URL: https://CRAN.R-project.org/package=dplyr
- **Lang, D. T., & Kalogeropoulos, K. (2022).** *RJSONIO: JSON for R.* URL: https://CRAN.R-project.org/package=jsonlite
- **Ooms, J. (2022).** *httr: Tools for Working with URLs and HTTP.* URL: https://CRAN.R-project.org/package=httr
- **Kahle, D., & Wickham, H. (2022).** *GGplot2: Elegant Graphics for Data Analysis.* URL: https://CRAN.R-project.org/package=ggplot2
- **Poppler Developers (2022).** *pdftools: Text Extraction, Rendering and Converting of PDF Documents.* URL: https://CRAN.R-project.org/package=pdftools
- **Smith, R. (2007).** *An Overview of the Tesseract OCR Engine.* In Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007), pp. 629–633. DOI: 10.1109/ICDAR.2007.4376991
- **LMStudio Community. (2023).** *Llama-3-Groq-8B-Tool-Use-GGUF.* URL: https://www.lmstudio-community/models/llama-3-groq-8b
- **OpenAI. (2022).** *GPT-3: Language Models are Few-Shot Learners.* URL: https://openai.com/research/gpt-3