Automated Loan Servicing Documentation

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Rudolph Schneider

Spring, 2023

Rosanne Vrugtman, Department of Computer Science

ABSTRACT

The Mortgage and Loan Department at Wells Fargo headquarters in Charlotte, North Carolina needed a system to automate the payment priority data on their service agreement records. To fix this issue, I built an application to automatically parse through the different mortgage deals and their respective service agreements and extract data concerning prioritization of payments on said mortgage deals. The application was built entirely in Python and employed a number of coding libraries to help me explore Wells Fargo's file system, as well as read in the service agreements and write extracted data to a separate file. Regular expressions and breadth-first searching was used extensively throughout my application to help me locate both the service agreements themselves and the required information within the agreements. The completed application was able to find and extract all required information from a given service agreement at an average of five seconds per deal at an accuracy of around 98%, greatly expediting a process that originally took several minutes to complete per deal. Further improvements to my application could give it a much need user interface to assist with its usability, as well as improvements on falsely flagged service agreements extracted by the application.

1. INTRODUCTION

Payment priority brackets are stipulations on a loan that describe in which order a loan should be paid back. For example, interest on the loan must be paid back in full before the principle of a loan can be paid. However, for the multi-million, or sometimes multi-billion dollar loans held by Wells Fargo, these payment priority brackets are exceedingly complex. A single deal can have as many as 32 different brackets, a level of complexity that made it increasingly difficult to document these brackets, especially when new brackets are created each year. For years employees within the Mortgage and Loan Department at Wells Fargo would undergo the yearly task of creating excel sheets listing the payment priority brackets of each of their thousands of loans they held. Employees were forced to read through the loan's entire service agreement, which is a 400-500 page document describing every detail of the loan, in order to locate the section dictating the payment priority bracket, and record its contents onto a spreadsheet.

Every step in this process has an asterisk. Sometimes service agreements are split into separate parts or located dozens of files deep within a directory. Which section in the service agreement contains the payment priority brackets is never consistent. The way the brackets are represented (e.g., lists, enumerated, if/then statements) change constantly. Wording for payment priority brackets are confusing and also vary wildly. All these inconsistencies resulted in an employee task that was not only boring and monotonous, but frustrating as well.

I automated this process to relieve employees of a task that wasted human resources, and nobody wanted to perform. I had the unique problem of creating a flexible program that both accounted for the issues in finding brackets and ensured that any brackets added in the future would be caught.

2. RELATED WORKS

The application I created at Wells Fargo falls under the branch of Robotic Process Automation. RPA is an increasingly popular mechanism to build automation on top existing computer systems in order to complete typically human tasks with robotic levels of precision and efficiency (Asquith et. al., 2019).

One such application of RPA to the field of auditing was proposed by Moffit, et. al. (2018) to complete more repetitive, trivial tasks during an audit to free human auditors to complete higher level tasks. This approach to RPA is very similar to mine, as it applies RPA to more monotonous tasks that can be applied to time-intensive jobs and completed without any level of machine learning or artificial intelligence. However, since auditing is such a robust process, not all tasks can be solved this way. Thus, this approach differs from my work at Wells Fargo as my role was confined to automating the process of finding priority brackets from start to finish.

Another case study on the use of RPA in the banking industry attempted the application of RPA to a broad array of banking-business related applications. Not only was this study by Romao et. al. (2019) more broad-reaching than mine, but also included uses of AI for automation. This approach makes sense for undertaking tasks larger and more complex than just documenting priority payment brackets, but is also far riskier. I considered using AI when I first started building my application, but ultimately decided against it due to time constraints and the risk of AI failing at tasks if I did not train my application properly.

3. PROJECT DESIGN

Wells Fargo divides the content of their different deals and holdings into several different directories. These directories each contain a large number of subdirectories that represent individual loans held by Wells Fargo. The loan subdirectories contain many additional subdirectories and other files, the service agreement being among them. It was my task to first locate the service agreement PDF file within these directories, then process the service agreement to find and extract the payment priority brackets within them. I used Python scripts for both tasks, as it was my preferred language and easily available to me as an intern.

I first familiarized myself with the language and layout of the service agreements I would be processing, along with the file systems that contained the deals. The language of these service agreements varied quite a bit from deal to deal, but there is a general format that almost all deals followed that allowed them to be processed for their payment priority brackets without machine learning. For example, all brackets are enumerated with either cardinal numbers, roman numerals, or ordinal numbers, and thus can be evaluated through predetermined regular expressions.

I first focused on the PDF processing, as this was the most important function of my project. I opted to use an open-source PDF reading library to make the documents digestible. The document first processed page by page and concatenated into a single, large string, as this made it very easy to work with. From this giant string I was able to locate the table of contents in the first few pages, and then isolate sections that might contain the payment priority brackets. This was done through regular expressions, by looking for keywords in section headers that were likely to have priority brackets.

However, an issue I faced with almost every PDF I ingested was that letters or entire words would be garbled when taken in by the reader. The issue lay with the PDF format itself, not the reader, so I had to design a work-around. I was forced to format every regular expression I looked for to account for possible errors, especially erroneously added spaces, periods, or deleted letters. The mechanism I used to search for the section that contained the required brackets was similar to a human's approach; I separated sections that might contain brackets, saved their page number, and looked at that page number to see if it contained language that pertained to enumerating brackets. Here I faced another issue, that many sections' page numbers were listed incorrectly. To account for this, I had to search several pages before or after the section to ensure I did not miss the section. Once I found the correct section, I separated the brackets using further regular expressions. The method of enumeration for the priority brackets varied from document to document, but luckily could be separated through regular expressions. Last, I extracted the brackets from my script and wrote them onto a separate spreadsheet file.

Once I completed the bracket finding mechanism, the next step was finding the deal's PDF file within Wells Fargo's directories. To accomplish this, I created a simple breadth first search script designed to search through a desired directory. The script would then check if any given file matched regular expressions that pertain to service agreement PDFs. If so, it opened that file and ran it through the bracket processing script repeatedly until it produced a single spreadsheet containing every service agreement's brackets of separate lines.

I designed my application for Wells Fargo employees within their Service and Loans department who had little to no experience with coding, so I wanted to make it easy to set up and use. I limited myself on how many libraries I needed to install so employees could download and set up my application quickly and hassle-free. I also made the functionality of my application come down to a single command line instruction, my main script followed by the path of the directory of interest, to further simplify things for the end users. Additionally, because my application was built out entirely in Python, only Python would have to be installed on the user's machine for my program to run.

4. RESULTS

The program I created was used by the Service and Loans Department of Wells Fargo in Charlotte to drastically decrease the time it took to extract payment priority bracket details for their mortgage loan holdings. Employees must find these brackets on both new and old deals, resulting in thousands of service agreements being evaluated or reevaluated every year. As preparation for my project, I preformed this evaluation manually on several different service agreements. On average, this took me around 10 minutes, not including the time theoretically required to search for the desired service agreement within a Wells Fargo directory, although I imagine a seasoned employee would be able to do this process in around half the time. In trial runs, my application extracted all required information in a directory full of service agreements at an average of about five seconds per service agreements. Most of this time was actually spent just searching for service agreements, and an isolated service agreement could be processed in around a second. Trial tests also showed that if a document was a service agreement, it was correctly identified and had its contents correctly extracted 98% of the time.

It should be noted that my program had a high false positive rate; that is, it processed many documents that were not actually loan servicing agreements. This was by design as falsely processed documents added little time and added harmless additional lines to the spreadsheet, in exchange for a low chance of missing actual service agreements. From the number of service agreements processed per year, I estimated the amount of time I saved in automating this process to be around 400 hours per year. At an average salary of 80k a year, this would save Wells Fargo around $16,000 a year.

5. CONCLUSION

The program I created utilized RPA techniques and regular expressions to properly document loan servicing documents for Wells Fargo in a fraction of the time when compared to traditional, manual methods of documentation. I found workarounds for formatting issues in the documents, simplified my design to account for useability, and trained employees on the use of my program. Through this project I eliminated a monotonous job for Wells Fargo workers, while improving both my technical and social skills. In the end, my work was able to save Wells Fargo an estimated hundreds of hours of productivity and thousands of dollars in wages.

6. FUTURE WORK

My project could be further improved by increasing ease of use and functionality. I was unable to implement a UI for my program during my internship, which would have greatly simplified the process of running my application. Since the end users have little technical background, using command-line arguments to run my program may still be confusing for them.

Additionally, I would have liked to have improved the high false positive rate of my program. Though it does little to affect the end product, it would make the output look far cleaner, and would be relatively simple to implement by tweaking the regular expressions that look for the servicing agreement PDFs.

There is also the possibility of adapting my project to other documentation tasks on other documents or information, not just loan servicing agreements. Given more time, I could refactor my code to find different kinds of data for servicing agreements or other financial documents.

REFERENCES

Kevin C. Moffitt, Andrea M. Rozario, Miklos A. Vasarhelyi. 2018. Robotic Process Automation for Auditing. *Journal of Emerging Technologies in Accounting* 1 July 2018; 15 (1): 1–10. https://doi.org/10.2308/jeta-10589

M. Romao, J. Costa and C. J. Costa. 2019. "Robotic Process Automation: A Case Study in the Banking Industry," *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*, Coimbra, Portugal, 2019, pp. 1-6, doi: 10.23919/CISTI.2019.8760733.