Capital One:
Letter Identification Using OCR

Team Members: Philip Foster, Andrew Mollenkamp, Austin Kirkpatrick, Jacob Camacho
UTD Advisors: Arunachalam Saravanan
Corporate Sponsors: Christopher Meyers, Roshni Rao, Enu Mittal

Abstract: Capital One receives thousands requests via mail every month from customers requesting to change data listed on their credit report. Presently, employees must scan these letters into an electronic database, however they must still manually read these letters to extract important information such as the customer's name, address, SSN, account numbers, etc. The proposed system will ingest scanned letters and process them using OCR and Natural Language Processing technology to extract relevant customer details and add them to the database, as well as attempt to detect if the request is frivolous. This will save Capital One employees time and increase the number of letters Capital One can process without needing to hire additional staff.

Table of Contents

Executive Summary

Capital One receives thousands requests via mail every month from customers requesting to change data listed on their credit report. Presently, employees must scan these letters into an electronic database, however they must still manually read these letters to extract important information such as the customer's name, address, SSN, account numbers, etc. The proposed system will ingest scanned letters and process them using OCR and Natural Language Processing technology to extract relevant customer details and add them to the database, as well as attempt to detect if the request is frivolous. This will save Capital One employees time and increase the number of letters Capital One can process without needing to hire additional staff.

Our UTD design team will build a Java application throughout the Fall 2018 semester, ideally communicating with Capital One members once a week for feedback and insurance of the desired product. Since the project is small enough, each UTD member will participate in all aspects of the project, although each member will have a specific emphasis on one of those aspects. Constant feedback and the lack of hard lines between key roles of the team should help to mitigate risks of the project and overdependence on any particular members. Resources used to communicate and create the project are free or supplied by UTD. All resources used in the project are open-source, avoiding costs and licensing fees.

This system will be designed to be easily expandable to address additional issues as they arise. The system will be scalable, which will allow multiple instances of the software to run on different machines at the same time. This will increase the maximum number of documents and requests that can be handled, and decrease the total amount of processing time.

Introduction

Each month, Capital One receives approximately 10,000 request letters via mail from customers to update information on their credit reports. Presently, these letters are read and processed manually by approximately 70 employees in the Credit Bureau Disputes Team. At present, the team's process for processing these letters involves scanning the paper letters and uploading them to a database. Employees then must read each letter and extract relevant fields such as customer name, address, SSN, etc. and enter them into a database by hand. This is time-consuming and prone to human errors. After entering the data, an employee must investigate the case to determine whether or not the request should be honored or rejected.

The Credit Bureau Disputes Team wants to automate part of this process by having an automated system read the letter to extract customer information and enter it into the database. This will save team members a significant amount of time by automating part of their current workflow. We are not aware of any alternative software solutions that exist to solve this problem.

Discussion

The core of the project will be written using Java and the Spring Framework. This gives us a good foundation which can easily be extended later on. This application will be placed in a Docker container, which will give team members, as well as Capital One, a consistent image that can be easily started and stopped, as well as replicated (using Kubernetes, or a similar tool) easily for scalability. We will be using Postgres for primary data storage, as well as Elasticsearch to store the converted text for fast searching.

To start, there will be an API where users can make requests to query the system and request new documents be ingested. This will be done with Spring Web - a part of the Spring Framework, which makes it easy to create RESTful APIs.

The document ingest subsystem will be responsible for reading new letters and processing them to extract relevant information. The ingest subsystem will be capable of accepting either single documents in .pdf, .png, or .jpg formats, and bulk ingest requests can be performed by uploading multiple documents in the supported file formats in a zip file. The system will then extract the text in the images using the Tesseract OCR library. Once the document has been converted to text, the system will begin processing this text to extract key customer information, such as account number, address, and name. Additionally, the system will sort documents into queues based on dispute characteristics, and calculate a probability that the document was created from a template.

One of the primary goals of this project is automated detection and extraction of key pieces of customer data from the letters. These fields are Account number, Address, SSN, Date of Birth, Date of Letter, Stamp date, and keywords. Each of these are unique and will require their own methods for detection. Extraction of the SSN is perhaps the most straightforward of all the above fields. SSN numbers have a unique format that can be picked up by a simple regular expression. Customer account numbers should also be fairly straightforward to extract with a simple regular expression, as they are simply a 16-digit number. To extract dates, we will use the “natty” library (http://natty.joestelmach.com/). This library provides a powerful way to parse dates in many different formats, and is free, open source software. Of course, there are several dates that we need to parse -- Customer date of birth, date of letter, and stamp date. The library will not be able to differentiate these dates on its own, however it should not be difficult to tell by a simple comparison. Customer date of birth will be the earliest of the three, followed by the date the letter was written, and finally the stamp date will be the latest chronologically. Address extraction is the most difficult, as there are many different address formats with varying amounts of information. To solve this problem, we have trained a custom machine-learning model compatible with the OpenNLP `NameFinderME` class. The dataset to train this model was taken

from a github repository (https://github.com/onethinglab/address_extraction/tree/master/dataset) and re-formatted using a custom python script (https://github.com/philipfoster/reformat-address-data). This result gives decent accuracy, and handles many edge-cases for address formats very well.

Template detection is a fairly complicated topic that will require a bit of statistical analysis to detect similarity to other documents in the database. Our first idea was to stem the document using PorterStemmer in the OpenNLP library, count the number of times each word appears, and compare the distributions between the document we're checking and every other document in the database. While this is promising theoretically, this would likely be far too slow to be practical. After a bit more research, we came across the Simhash algorithm. This allows us to compute a 128-bit "fingerprint" for each document and store it in the database alongside the original text. The unique property for Simhash that makes it particularly useful is that the hash digest will be similar for documents that have similar content. To compute a "difference score" between any two documents, we take the simhash digest of each document and `xor` them together, then count the number of 1s in the result. We then take that count and divide it by 128, to get the difference score as a percentage of the entire document. To convert that to a similarity score, we simply compute `1 - difference_score`. To check if the document is a template, we compare the simhash digest of the document we're investigating against the digest of each previously scanned document in the database, and increment a count if the similarity score is above a certain threshold (say, 75% similarity). If, at the end, the count is above a certain number of documents, then we can infer that the document we are looking at is based off of a template document.

In a test implementation of the simhash and comparison algorithm, we found that computing the similarity score takes approximately 0.04 ms per comparison. Assuming 600,000 documents in the database, we can calculate that it would take approximately 24 seconds to exhaustively compare the target document against every other document in the database. If this proves to be too long, there is potential to significantly improve this by dividing the work across multiple work threads. Other performance improvements can be had by randomly selecting a percentage of the documents and only comparing against them. This should give a good estimate by taking advantage of the Central Limit Theorem, assuming the percentage is sufficiently large.

The system will also take the documents and sort them into queues based on dispute characteristics. Our first thought was that we could have a list of keywords that indicate which queue(s) a document should belong to. While this would be fast and simple to implement, we have some concerns about the accuracy of such a design. After much thought, this task seems to be much better suited to a machine-learning approach. Fortunately, the OpenNLP library we are already using in the project has built-in tools for document categorization which should solve

the problem. Unfortunately, we don't have the data needed to train a model to solve this problem. This data can be acquired, however, with some time and a temporary, low-impact modification to the user workflow. To obtain the data, Capital One employees who process the letters will manually sort the letters into queues for a few months. After a month, they will have approximately 10,000 examples that can be used to train a model. Once this model is generated, we will test it for accuracy. If the accuracy is satisfactory, then the workflow modification can be removed, and the algorithm can automate this task into the future. If the accuracy is not satisfactory, then we can extend the time another month to gather more data, which should help further improve the accuracy. We will provide documentation on the process for manually sorting into queues, as well as steps for training the model and applying it to the project.

The API is split up into 3 parts: document ingest, data querying, and updating data. (complete documentation available at https://philipfoster.github.io/CapitalOne-OCR-Project/) The ingest system works asynchronously by creating a job that will be processed as soon as possible. This was done to prevent a potential problem that would occur if an upstream web server were to reset the connection for a long-running batch operation. Clients will be able to query the status of a job, and retrieve data about the completed job when results are available. After documents have been processed by the system, clients will be able to get the data extracted by the system, as well as converted text and the original images. If, for some reason, the system incorrectly extracts some data, or misses some important data, the person reviewing the document will be able to manually update the incorrect data.

All of the data for the project will be held in a PostgreSQL database. The schema is centered around the document table, which contains the data that has been extracted from the document, as well as references to other data. The document_text table will contain the text pulled out of the image from the Tesseract OCR software. The queues table contains a list of possible queues that a document can be sorted into. An entry in the document_queues_relation table indicates that a document is a member of a certain queue. The document_images table will contain the raw scanned images to be processed. This is referenced by a job task, which contains an "intent" for the image to be processed at some point when a server is available. When an available server claims a job, it will create a row in the job_assignments table. This marks a job as taken, and removes it from the task queue. This does create a potential issue, however: If a server instance crashes before it can complete a job, then the document will never be fully processed, and will not be added back into the queue for another instance to process. To handle this case, a script will occasionally run that checks jobs currently being processed. If the current time minus the started time is greater than six standard deviations from the average processing time, then the script will assume that the server has crashed, and will un-assign this task, which will put it back in the queue. The script will additionally check the server in question, and restart it if necessary.

Resources

- Personal computers of each of the members of the team
- Slack:
    - https://slack.com/
- Cisco Webex Meetings (Provided by UTD)
- IntelliJ IDE:
    - https://www.jetbrains.com/idea/
- GitHub:
    - https://github.com/philipfoster/CapitalOne-OCR-Project
- Tesseract OCR (Optical Character Recognition) Open Source Project:
    - https://opensource.google.com/projects/tesseract
- Java JNA wrapper for Tesseract OCR API:
    - https://github.com/nguyenq/tess4j
- Fake letters (Provided by Capital One) for testing
- OpenNLP
    - https://opennlp.apache.org/
    - Open source NLP library sponsored by the Apache foundation.
- Spring framework
    - https://spring.io/
- Trello
    - https://trello.com

Key Roles

All members will be working in all aspects of the project, but each member will have an emphasis on a particular aspect of the project.

Jacob Camacho - Databases: This role emphasizes creating and maintaining the structure of the project's SQL database as well as ensuring that the information gathered from each letter is properly formatted to fit the database schema.

Philip Foster - Software designer: This role emphasizes decisions over the the overall structure of the project and what external programs and software are to be used.

Austin Kirkpatrick - Documentation: This role emphasizes maintaining a healthy amount of documentation in the code through comments and out of the code through the readme in GitHub. This will help team members and future developers be able to pick up another member's work with as little difficulty as possible.

Andrew Mollenkamp - Testing: This role emphasizes testing the project as it develops in various ways to ensure functionality and to reduce the buildup and oversight of bugs. All bugs of course will be marked in GitHub and through communication channels to the team.

Communication Plan

We plan to have weekly team meetings at minimum with biweekly meetings being preferred. These meetings will be each Tuesday and Thursday at noon, and will last as long as the team feels is necessary at that point in time. Our advisor will join in on these meetings occasionally as often as he feels is necessary. Additionally, we will meet with the corporate sponsor weekly on Friday at 3pm. Team meetings will be held virtually via WebEx unless the team agrees that an in-person meeting is necessary. Meetings with the corporate sponsor will take place virtually using a Zoom link provided via email by Capital One. In this case, the team will work together to determ*ine a suitable location to meet at. We have created a Slack group for general communication which does not require a meeting. Members can be in near-constant communication whenever it is necessary.

Risk Analysis / Contingency Plan

There are several risks present for this project.

Team Member is Unable to Work

There are several extreme circumstances, such as severe or prolonged illness, which may prevent a team member from being able to work on the project. If this happens, the affected member will communicate this with the rest of the group so that the group can plan to complete any work that the affected person cannot do. This will ensure that the project remains on-schedule.

Team is Unable to Complete the Project by Delivery Date

One of the goals of this project is to create a product that is well documented and have a high level of readability. In the event that the project is unfinished by the delivery date, our team hopes to give at least a partially finished product that another development team can easily continue the remaining development with as little transitionary difficulties as possible.

The Tesseract OCR Proves Unviable for the Project

From our research, Google's Tesseract OCR API is the only reputable open-source product. In the event that Tesseract proves unviable for the project, other APIs will be further researched although the expectation is that other APIs will be subscription-based. If Capital One does not agree to purchasing/using another software, our goal is to deliver as much a finished product as possible, with functionality in all other parts of our project.

Costs

We do not anticipate any costs for this project. This is a purely software project, so there are no hardware requirements. Any software dependencies we use will be exclusively free and/or open source, per Capital One's request. Team members are free to use any developer tools that they already have to create the project. Any proprietary software that we use (i.e. WebEx) is either provided by the university, has a suitable free tier available, or has free student licenses available.

Timetable

Project Start

September 5th

Project Proposal Completed

September 23rd

Begin Implementation

October 5th

Project Proposal Due

October 5th

Poster/Slides Due

December 10th

Project End

December 13th

Evaluation

We will consider the project to be successful if the software functions to specification, and is able to reduce the amount of work required by humans to perform the task of processing request letters. We will create a Trello list to keep track of tasks and the team's progress. As bugs are discovered during development, we will create issues in the GitHub repository's bug tracker. This will allow us to make sure that we do not lose track of bugs, and gives us an objective measure of progress. Additionally, we will strive to keep a high level of test coverage, which will ensure code quality and help make sure that bugs do not go unnoticed.

Performance of the system will also be measured to ensure that the system can process requests in a reasonable amount of time and that the results of the queries are of high quality. The first measure we will use is the accuracy of the OCR software. This is important, because it serves as the basis for the rest of processing that the system will do. The next field we will use is the accuracy of field recognition, as this is the most important task that the system will perform. Finally, we will measure the speed with which the system can process a letter, as well as the maximum throughput that a single instance can process within a reasonable amount of time.

Conclusion

This project gives corporations, like Capital One, the ability to reduce mundane and labor intensive task by automating the recognition of important fields in customer complaint letters. While this project will not be able to fully automate the process, it will help to reduce the number of man-hours put into simply reading and parsing letters. With the use of our machine learning and OCR software, the team which processes customer complaint letters could be reduced or their time better spent on other activities, better increasing the efficiency of the organization.

Contact Information

Philip Foster:
Phone: (281) 740-9439
Email: philip@pfoster.me

Andrew Mollenkamp:
Phone: (469) 579-9524
Email: ajm151130@utdallas.edu

Austin Kirkpatrick:
Phone: (972) 983-8287
Email: abk150030@utdallas.edu

Jacob Camacho:
Phone: (214) 250-8946
Email: jbc140230@utdallas.edu

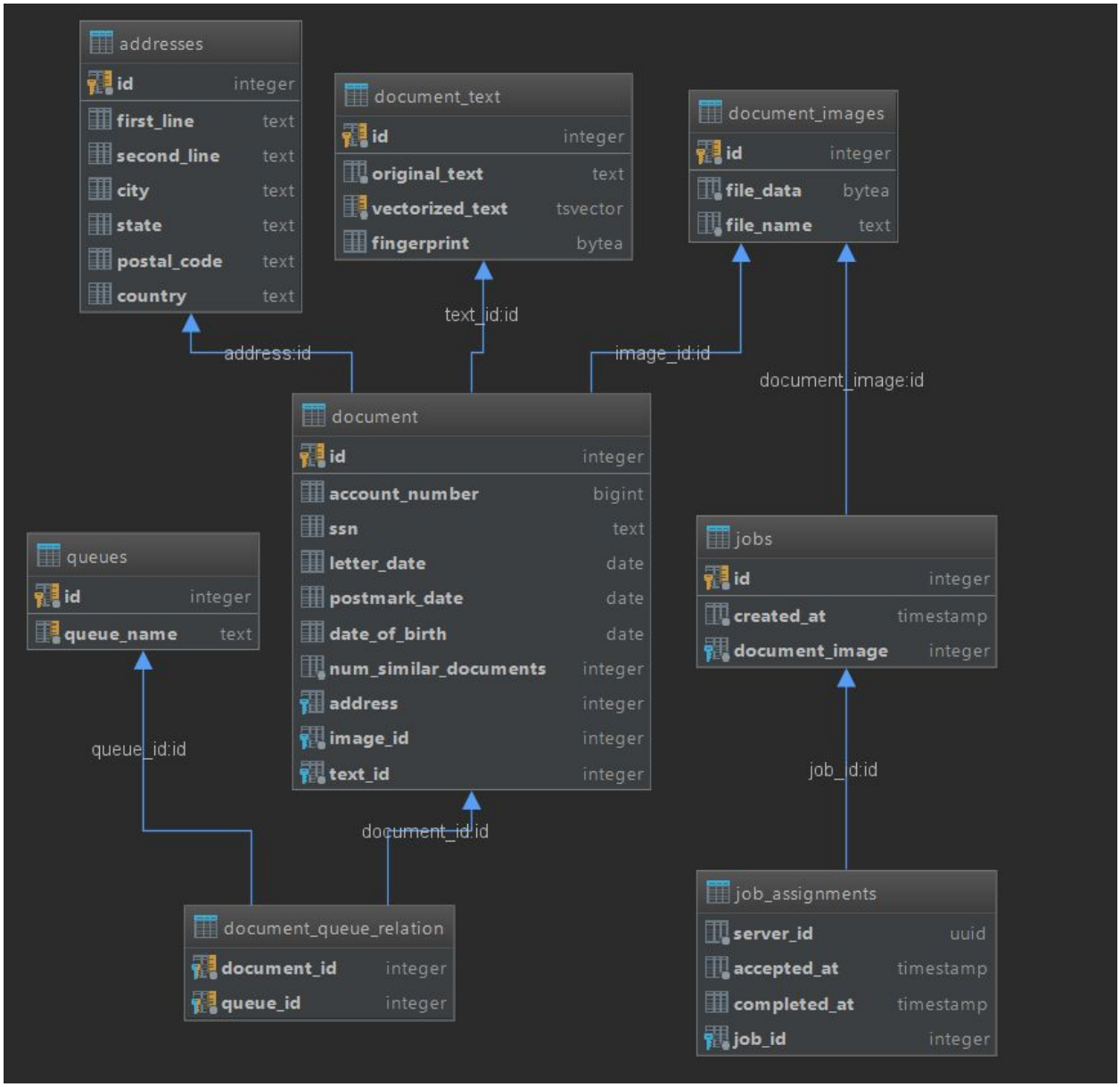
Arunachalam Saravanan:
Phone: (469) 395 9357
Email: axs170081@utdallas.edu

Sources

- Near-Duplicate detection:
  - https://moz.com/devblog/near-duplicate-detection/
- Postgres full-text search:
  - http://rachbelaid.com/postgres-full-text-search-is-good-enough/
- Statistical error detection:
  - https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule
- OpenAPI Specification
  - https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.0.md

Appendix

Database Schema Diagram



Approval

_____ Philip Foster_____
Signature of Team Member Print Name

_____
Date

_____ Andrew Mollenkamp_____
Signature of Team Member Print Name

_____
Date

_____ Austin Kirkpatrick_____
Signature of Team Member Print Name

_____
Date

_____ Jacob Camacho_____
Signature of Team Member Print Name

_____
Date

_____ Arunachalam Saravanan_____
Signature of Faculty Advisor Print Name

_____
Date

_____ _____
Signature of Company Mentor Print Name

_____
Date