Civera MassCourts

04.07.2021

Ammar Ahmad
Sherry Courington
Megan Mastrorilli
Shihab Karim
Serra Jung

Project Introduction

Civera is a software company seeking to liberate public data by making it easier to access and understand. The state of Massachusetts court system has a website, MassCourts.org, containing its public data. However, this website is unorganized and difficult to navigate, rendering it, and the data, useless to anyone trying to access it.

Civera has scraped the entire data from the official website into a database. They have designed an alternative website using that database of court data. However, to complete this project, they need the data to be in a standardized and clean form.

Hence, we were tasked with standardising the raw data from the MassCourts website into a new more readable dataset. Our main task is to standardize the database and perform analysis on the court data.

Dataset

The data set is client's raw data for MA housing court dockets. It includes four tables that are stored in a MySQL database hosted on a server.

Working with Data

The dataset that we have for this project is huge (in GBs). There are four tables each with millions of rows. We are using the following ways to interact with the data

- Connecting to the data server using MySQL workbench and writing queries on MySQL workbench.
- Connecting to the database server using python and reading the data using queries. We sometimes retrieve the result of the query in chunks since the size of the data is huge.
- Downloaded the tables as a csv file and read the csv file using python.

Tasks and Progress

Task	Status	Comments
Converting PHP regex to Python	Complete	Current code outputs an actor from a given match

		(Github existing code is hardcoded for the judge actor), might need to be modified/hardcoded for certain actors. Can modify to output action
Normalizing the database (fill missing action/actor values)	In Progress	
Analysis on Normalized Database	To do	

Methodology for Normalizing the Database

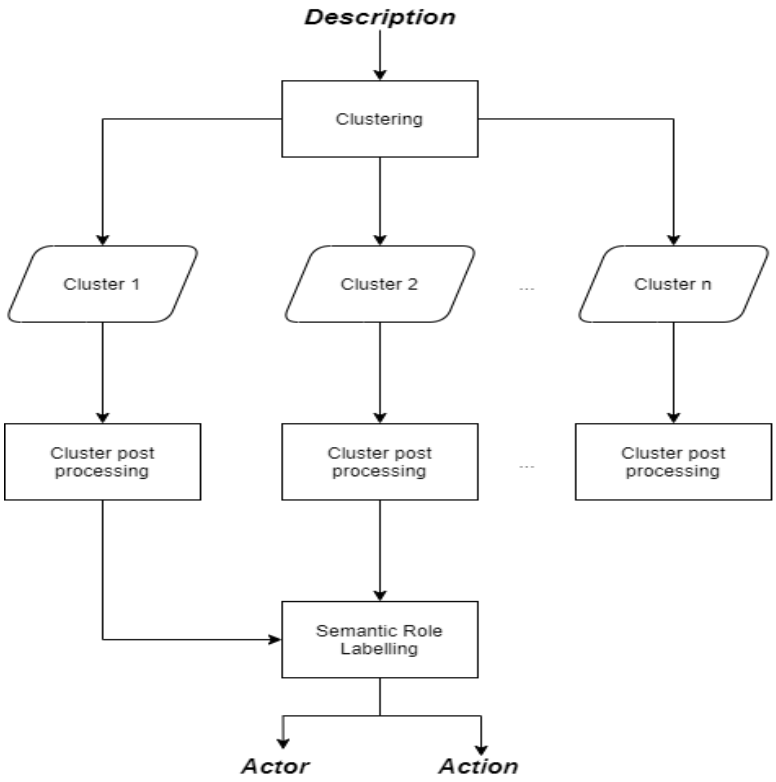
Each case in the raw dataset has different case actions that happened at different times. Moreover, each case has a case description. Using these case descriptions, we can infer the case action and the case actor. We were tasked to extract the case action (what happened) and case actor (who completed the action) for every entry.

The main task in normalizing the database is to fill in the missing action and actor values for the case actions. Each case action has a one line description from which we can extract or infer the case action and actor. Therefore, this becomes an information extraction task.

The client had done some of the extraction manually using regex. But since the data is so huge, we attempt to do this in an automated manner. To solve this problem, we first analyze the text and then try both supervised and unsupervised approaches based on our initial analysis of the data.

Unsupervised Approach

Our goal is to design the following pipeline:



In the unsupervised approach, we first cluster the text into different clusters or categories and then apply some approach to extract the actor and action from descriptions.

According to this pipeline we perform the following steps

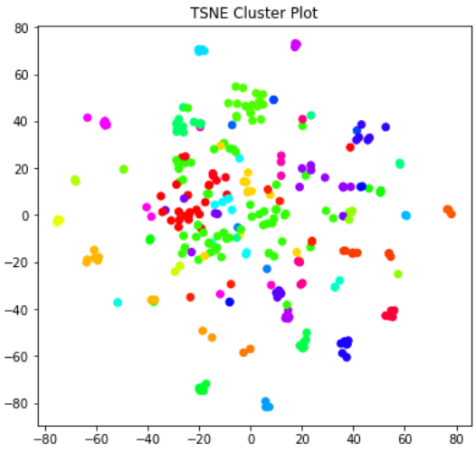
1. Text Pre-processing
2. Clustering the descriptions to different clusters
3. Cluster post processing
4. Semantic Role Labelling

Text Pre-processing

Each text description has to be converted to a set of numbers. So we tried the following methods to perform this conversion:

- TFIDF Features

These are features that are generated according to the count of each word in the complete dataset. The following TSNE Cluster Plot illustrates how these data points are similar:



- Glove Embeddings

Pre trained glove embeddings used to convert each text description into its equivalent representation.

- Fasttext Embeddings

These embeddings were trained by Facebook, We use these pretrained embeddings as well.

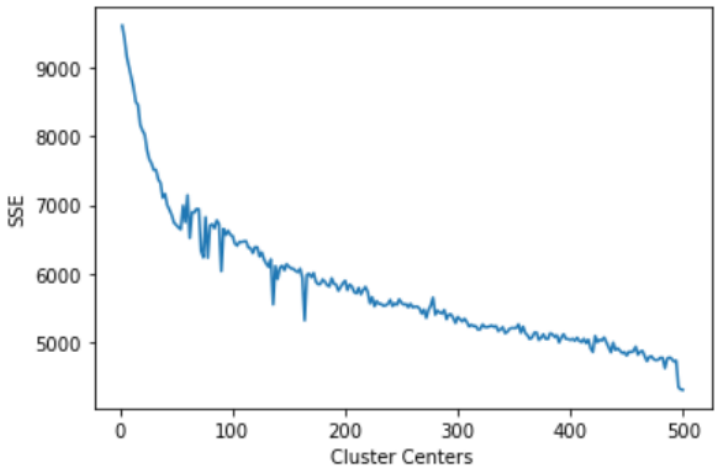
- POS (Parts of Speech) Features

These representations were made by converting each word into its part of speech.

Clustering

Our intuition is that all the case actions of the same kind should fall into the same cluster. Also we think that there will be many clusters which will be similar semantically, so we can use the same approach to extract the case actions from them.

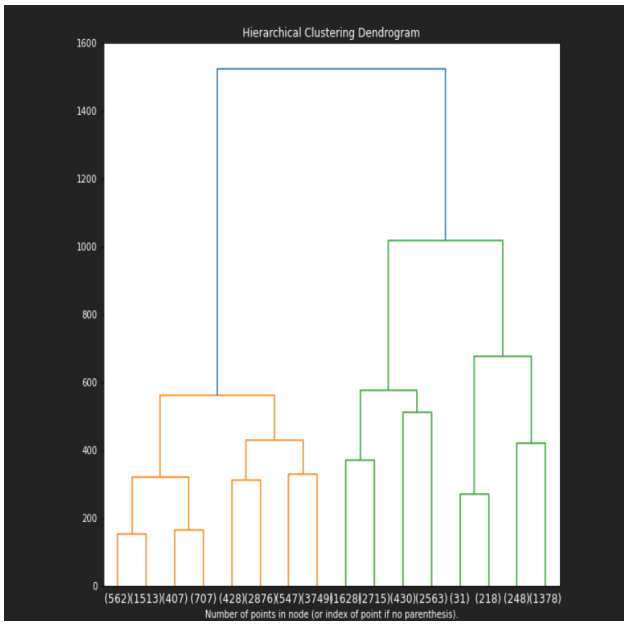
To perform clustering on these vectorized text descriptions, we utilized three different clustering algorithms: K Means clustering, Mini Batch K Means and Hierarchical Clustering. KMeans was time-intensive, even on GPUs and failed to produce the required elbow. We then transitioned to a different version of KMeans, the Mini Batch K Means and found the elbow with the number of clusters equal to 50. See graph below:



Choosing the optimum number of clusters

One of the typically large challenges in clustering is choosing the optimum number of clusters. We utilized the traditional method referred to as the elbow method. With this method we tried a numerical range of clusters and plotted the Sum Squared Error, SSE *(the difference between each observation and the mean of the cluster)* generated by each cluster on a graph. The elbow, or the break in the graph, indicates the point where it is no longer advantageous to create additional clusters.

When utilizing these 50 clusters did not yield the results we anticipated, we used Hierarchical Clustering to cluster similar objects into groups. These 16 groups, see graph below, allowed a better result in semantic labeling.



Cluster Post Processing

We observed that our clustering using different methods was very good but not perfect. So to improve the clustering results, we will add a cluster post processing step. In this step, we take all sentences in a cluster and exclude the sentences from the cluster that are different from the majority of sentences because theoretically we should have similar sentences in a cluster.

Extraction using Semantic Role Labelling

We saw that most clusters were in passive voice and identified some common passive voice patterns. To extract action and actor from these clusters of sentences, we are going to use an NLP approach called Semantic Role Labelling. We are going to use a state of the art pretrained machine learning model for this task.

Supervised Approach

In the supervised approach, we tried to perform classification on distinct case actions and case actors. We attempted to design models that predict what case action and actor each description belongs to.

Text Pre-processing

To do this we again had to convert the text descriptions to numbers using TFIDF features.

Models Tested

We tried the following models:

1. Linear Singular Value Decomposition on TFIDF Features
2. Multinomial Naive Bayes on TFIDF Features
3. Random Forest on TFIDF Features

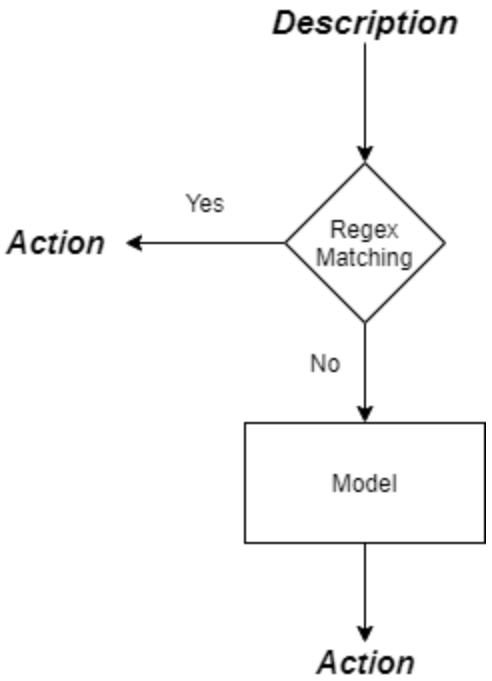
Imbalanced DataSet

One big challenge in training supervised models is the imbalanced dataset that we have. This means that there are very few examples of some case actions and actors while a very large number of examples for other case actions and actors. This imbalance is inevitable given the nature of the data. We first attempted to solve this problem by creating a custom balanced training set for our models, where for each distinct case action, we took 10% of its rows to create a training set. Assuming the remaining unclassified data had similar ratios of common and uncommon actions, we thought this would work. Another way we attempted to solve this data imbalance was by taking exactly 10 rows from each distinct case action to create a training set. However, both of these methods continued to result in a low accuracy score on all three models.

Final Methodology

Semi-Supervised/ Combined Approach

The following diagram illustrates the algorithmic approach our analyses led us to, given the accuracy of its output:



After trying all the above approaches, we came to the conclusion that a combined approach involving the regex and the supervised approach of a fine tuned text summarization transformer works the best. For some descriptions, it is easier to just match regex patterns to extract actions. These actions and actors are straightforward. We were also provided with some regex patterns that were already written for this task in PHP code. We converted those to python regex. A description is first matched with regex patterns and if a pattern does not match then the description is given to the trained model and the model outputs the action.

Running Parallel

Based on the massive raw volume of the data we are normalizing the database by running the same code at multiple places. To do this, we need to keep track of the data that each instance of the code is working on. Therefore, we created a table called “maintainer” in the normalized database to keep track. The whole process of normalizing the database involves reading the data from the database and writing back to it. In addition, it also

involves reading and writing from the maintainer table. The pipeline works on chunks of 100 rows. The steps taken by the code to normalize are the following

1. Get 100 'available' case action ids from the **maintainer** table.
2. Update these case action ids in the maintainer table to '**in progress**' .
3. Read the rows corresponding to these case action ids from the cdocs_case_action_index table.
4. Normalize the rows
5. Write the normalized rows to the NORMALIZED case_action_index table.
6. Update these case action ids in the maintainer table to **'done'.**

Performance Analysis

We analyzed the effectiveness of this approach by looking at its speed. We look at the individual components of this pipeline and mention the time taken by them on an individual row and on a chunk of 100 rows to get a better idea. Since we are running this code on google colab, all of the below analysis with respect to time is according to the hardware provided on a google colab notebook with a GPU.

Action

Component	**Time Taken on a chunk of 100 rows (seconds)**	**Time taken on a single row (seconds)**
Regex Extraction	0.07	0.0007
Extraction by ML model only (GPU)	4.80	0.48

It is also important to note that even though the regex is much faster than the model, the regex does not cover each and every case action. Hence, the model is used in conjunction with the regex.

Actor

Component	**Time Taken on a chunk of 100 rows (seconds)**	**Time taken on a single row (seconds)**
Regex Extraction	0.06	0.0006
Extraction by ML model only (GPU)	3.52	0.352
Getting Parties associated with the case	0.92	0.092

Actor Verification with Parties	0.08	0.008

It is also important to note that even though the regex is way faster than the model, the regex does not cover each and every case actor. Hence, the model is used in conjunction with the regex. Additionally, once a party is determined as the actor, it is verified with the parties associated with the case.

Querying the Database

We cannot accurately calculate the time taken to run the normalization pipeline on a chunk because we do not know how many rows will be normalized by the regex and for how many rows would the model be needed. The accurate time also depends on how many instances of this pipeline are being run at the same time parallel to each other. Each running instance will query the database and the queries can take different times based on how many queries are being run on a table. However we do provide an estimate of how long these took on average when 4-5 instances of the pipeline were running in parallel.

Components	**Time Taken on a chunk of 100 rows (seconds)**	**Time taken on a single row (seconds)**
Reading from MAINTAINER table	0.59	0.0059
Updating MAINTAINER table	8.29	0.0829
Reading from case action table	0.32	0.0032
Insert to case action normalized table	8.21	0.08
Updating MAINTAINER table	9.41	0.09

This table shows that the whole pipeline is slow due to the update and insert queries.

Limitations, Challenges and Lesson Learned

The immense size of raw data makes it difficult to try different approaches since it takes a very long time to process the complete dataset. To deal with this, we have tried using our approaches on smaller subsets of the data. However, since we sometimes are unable to perform a technique on all the data, there is a certain amount of guesswork involved which is eliminated when we test our approach on the whole dataset.

This was a very interesting project benefiting us as students, as well as the client. The dataset was immense and not cleaned or formatted, as are some datasets. We acquired a lot of hands-on experience experimenting with and utilizing a variety of AI models, as well as expertise in cleaning and pre-processing the datasets. The experience gained from working in a group and counting on others will benefit us post-graduation.

** *Writing to the SQL database slows the productivity of this pipeline. Hence, we are delivering the executable code with instructions for the client to continue normalization. Conveniently, the code provided interacts with the database this team was asked to write to and is approximately 2% complete as of this project's deadline.* ***Please follow the directions in the Read Me before utilizing it.***