Extracting a Knowledge Graph from Historical *Emigrant Industrial Savings Bank* Data

Stephen Balogh
Department of Computer Science
Courant Institute
New York University
`sgb334@nyu.edu`

Chun Ming Ho
Department of Computer Science
Courant Institute
New York University
`cmh531@nyu.edu`

Abstract

A dataset representing accounts and account-holders at the Emigrant Industrial Savings Bank, the oldest savings bank in New York City, presents a large amount of unstructured or semi-structured natural language content of substantial historical import. In this paper, we detail our efforts using machine-learned algorithms for information extraction in order to parse this data into a format where it can be queried and traversed as a graph. Our approach involved the use of the conditional random field (CRF) algorithm to label token sequences, and the creation of a formal context-free grammar to parse particular types of claims from resulting sequences of labels.

1 Problem setting

The project at hand concerns the recognition of meaningful named entities, and the parsing of specific semantic relationships between these entities, from a corpus representing the account records from New York City's Emigrant Industrial Savings Bank (henceforth "ESB" or "EISB"). As described by a guide accompanying the data, published by the New York Public Library:

> *The Emigrant Savings Bank was founded by officers of the Irish Emigrant Society as an outgrowth of the society's concern for the welfare of Irish immigrants. The bank's opening in 1850 coincided with the waves of immigration that followed in the wake of the Irish Famine. [...] From the start the bank's business was very closely related to the work of the Society. As a result many of the volumes in the collection are personal and family chronicles as well as records of banking transactions. Collectively these volumes rank with the richest sources to be found on either side of the Atlantic for the study of Irish emigration.* (6)

We were attracted to this dataset for two reasons: 1) it presents a real, yet bounded, challenge in terms of information extraction from natural language, and 2) the creation of a graph network of entities and relationships from this data would potentially be of significant use to the New York historical community. Indeed, our discovery of this dataset comes simply from a coincidental discussion with Nicholas Wolf,[1] a historian and librarian at New York University, who has been engaged in efforts along with other academics to manually extract structure from the dataset for over a year.

[1]`https://as.nyu.edu/faculty/nicholas-wolf.html`

2 Dataset

The actual ESB corpus consists of semi-structured data. In its current form, transmitted to the authors by Dr. Wolf, we have access to manually typed transcriptions of the original handwritten records, which present a small amount of some of data for each account. The structured data fields present within the corpus includes information on an account holder's occupation, year of account creation, as well as their name and immediate relatives' names.

Of primary relevance to this project is the column of largely unstructured data, referred to in the dataset simply as "remarks". Below are some representative examples of this "remarks" field:

1. `She Nat of Ferrymount, 6 miles from Mt Mellick, Queens, Ire - Arr Jul 1844 per Fairfield from LP - Fa in Ire John Henry, Mo dead Bridget Fahy, 4 Bros Pat'k, John & James in US, Martin in Ire, 3 Sis Ellen, Honora & ___ see 3989`
2. `Lived at the corner of Sherriff and Stanton St and 3 mos Since - Has Just Returned from Washington City - Has been to see Her Parents at Fredricksburgh Va - A Widow`
3. `Nat of Ire, His Son Thos is Nat of Ireland - Works with W R Dewhurst Co & Is Single - Mo dead Ann Collins`

It is possible to quickly get a sense of the rich semantic relationships encoded within the "remarks" field simply by glancing at the above examples. In terms of entities, we see mention of individuals, specific locations, trans-Atlantic ships, and companies. And of the relationships described, there are those concerning family and descent, emigration, births and deaths, and employee-employer connections, among others.

We have access to 25k rows of this data – though because some entries do not have any data in the "remarks" field, the number of entries containing meaningfully parse-able natural language may be closer to 20k.

3 Brief survey of related work

Dan Jurafsy and James Martin's "Speech and Language Processing" textbook chapter on information Extraction provides a very useful initial treatment of popular approaches in IE, and provides some analysis of many of the strategies that will be relevant to our project. (4) In particular, the chapter conceptualizes NLP information extraction systems in terms of a pipelines of successive processes, involving – for instance – multiple passes of high-accuracy taggers, which rely on catalogs of known entities, and which increase the performance of probabilistic models that tend to be applied subsequently within a pipeline.

Our focus on sequence models – and, in particular, on the conditional random field (CRF) algorithm – is partially informed by a paper published by developers at the New York Times, detailing their use of the algorithm for an information extraction task involving cooking recipes. (3)

Chiticariu, et al., in their 2013 paper, detail how many "top-of-the-line" knowledge extraction systems still rely on explicit rule-based parsing for performing a large amount of their work. Generally (as echoed by Jurafsky and Martin), only after rule-based approaches are exhausted are machine-learned classifiers typically used in these systems.

4 Sequence labeling

Upon receiving the data, we first inspected instances of the "remarks" field and compiled an initial set of types of claims that seemed to appear with enough regularity to be extracted reliably in our system.

We first created a set of broad "statement themes"; there are, for example, portions of the "remarks" text pertaining to the parents, siblings, and spousal relations of the account-holder; additionally, there are statements about where individuals are natives of, about the circumstances of emigration to the United States, about employment, and also meta-references (by the bank) that connect accounts to others.

From this general list of themes, we set about constructing labels to use with the goal of first simply being able to identify which portions of a "remarks" text map to which specific themes. We reasoned that once we had semantically categorized contiguous portions of the text, it would be easier to then extract specific types of relations, properties, and entities, all of which requiring careful token-level consideration.

4.1 Conditional Random Fields

For the task of partitioning semantically distinct themes from the input, and also the subsequent task of labeling tokens (within themes) with their class, we chose to use an implementation of linear-chain conditional random fields. The formulation of the algorithm is documented originally by Lafferty *et al.* (5) While it was evident that we would need some sort of sequence model, we chose CRF over a Hidden Markov Model for a couple reasons:

1. We had to assemble our training data from hand, and because the amount of labels we wanted the system to predict is high, we knew that training sparsity would be a big issue for us, and feature-based models tend to respond better in these circumstances.
2. Additionally, we wanted the ability to extract features from across the entire "remarks" field, and did not want to constrain our model only to signals occurring within small window of tokens.

We used the Python `sklearn-crfsuite` implementation of CRF,[2] and built our model and feature extractors on top of it. We used a set of hyper-parameters deemed reasonable by the package documentation; model learning was performed using the default implementation of the L-BFSG gradient descent algorithm with a maximum of 1000 iterations, and the model considered all possible state-to-state transitions.

For our task, we ended up training two different CRF models were on distinct sets of labels:

1. A "statement theme" model, which maps (generally contiguous) sequences of tokens into broad thematic categories (with 17 possible labels) – for example, a section of tokens pertaining to the account holder's siblings would get the *fam:siblings* tag.
2. A tagger that maps tokens into symbols representing entities, subjects, and simple relationships (e.g., a token might be identified as containing a portion of a person's name).

The "token label" model was trained with access to statement labels. This is discussed in following sections.

Table 1 depicts a sample set of annotated tokens.

CRF feature engineering We wrote a series of feature extraction functions, the vast majority of which were used during training for both of the CRF models.

For input to the models, the content of the "remarks" field of a record was first tokenized into a list by splitting the original string on white space and non-alphanumeric characters.

For each token in an input sequence, we then assembled a vector containing features representing word shape, suffix / prefix bigrams and trigrams, presence of some known n-grams with unambiguous meanings (e.g. "Ire", a common abbreviation for "Ireland"); also, the token's relative position in the sequence, presence of non-alphabetic characters, whether or not numbers are within realistic year ranges, and others. A couple features encoded syntactic properties, such as if a token falls within open parentheses or open brackets.

For each token, these features were extracted on the present token, as well as for all tokens within windows of size 3 to the left and right.

Two-passes of CRF As already touched on, we decided to train two CRF models with separate sets of parameters, with the intent of using them in consecutive passes of labeling on an input sequence of "remarks" tokens. One motivation behind this was that we were not sure how well CRF would work

[2] `https://sklearn-crfsuite.readthedocs.io/en/latest/`

Table 1: Example of CRF labels

	Token	Statement/Theme Label (Pass 1)	Token Label (Pass 2)
0	<START>	START	START
1	Nat	subj:native-of	t:rel:IS_NATIVE_OF
2	of	subj:native-of	t:rel:IS_NATIVE_OF
3	Portadown	subj:native-of	t:location:NAME
4	,	subj:native-of	t:location:NAME
5	Co	subj:native-of	t:location:NAME
6	Armagh	subj:native-of	t:location:NAME
7	,	subj:native-of	t:location:NAME
8	Ire	subj:native-of	t:location:NAME
9	-	delimiter:thematic	delimiter:thematic
10	Arr	subj:emigration-event	t:emigration:ARRIVED
11	NY	subj:emigration-event	t:location:NAME
12	Sep	subj:emigration-event	t:time:MONTH
13	22	subj:emigration-event	t:time:DATE
14	1849	subj:emigration-event	t:time:YEAR
15	per	subj:emigration-event	t:emigration:VIA
16	ship	subj:emigration-event	t:emigration:VIA
17	Reinzie	subj:emigration-event	t:emigration:VESSEL
18	fr	subj:emigration-event	t:emigration:VESSEL_HAS_ORIGIN
19	Belfast	subj:emigration-event	t:location:NAME
20	-	delimiter:thematic	delimiter:thematic
21	Fa	fam:parents	t:person:FATHER
22	in	fam:parents	t:person:LOCATED_IN
23	Stephensville	fam:parents	t:location:NAME
24	,	fam:parents	t:location:NAME
25	Sullivan	fam:parents	t:location:NAME
...	...	...	...

for purposes of token-level annotation, given our small amount of training examples, and that having a more general "statement" level CRF model would at least provide some value.

Another motivation was that the presence of statement labels, if sufficiently accurate, could be an important signal for token-level CRF predictions. For instance, knowing that a token is in a segment of the "remarks" with the statement label `subj:native-of`, versus `person:brothers`, might help the model correctly identify a token "Kilkenny" as being a the name of a place rather than a male sibling (even though they otherwise likely share many of the same features).

CRF label creation Labels for the two CRF models were created iteratively, mostly during the process of annotating our training data. We started with a set of labels representing the main types of statements, and slowly modified and added to these after annotating an initial batch of data. We did likewise for creating labels for individual tokens.

At final count, the label space size for the "statement theme" CRF model is 17, whereas the size of the token-level label space is 66.

Compiling training data Initially, we simply selected records to be annotated for use as training data by pulling them from the front of the list of 25k entries. All records were unlabeled when provided to us.

Later, we started selecting records randomly for annotation, drawing from the entire span of the corpus – and also in a directed fashion, as in cases where we noticed certain phrases or constructs that had previously not been seen, or which were underrepresented in the training corpus.

We ended up using 81 labeled "remarks" sections, spanning 1640 tokens, as training data; the same data was used to train both CRF models.

5 Sequence interpretation

Our goal, in terms of interpretation of the original "remarks" data, is to be able to convert the original natural language text into a data structure that clearly expresses graph edges / nodes from a set of potentially detectable entities and relationships.

For example, given the statement:

```
Nat of London, Eng - Arr NY per ship Orinthian fr London - Par dead Fa Jno,
Mo Sarah ...
```

... we want to extract a data structure that clearly indicates that our graph needs:

1. A node representing the location "London, Eng"
2. An edge connecting the account holder's node to the London node
3. A node for the ship Orinthian, and an edge connecting it to the location London
4. Two nodes for the parents, each containing their names and whether they are living/deceased (and two edges connecting the account holder to their mother and father)

We experimented with two methods of interpreting sequences of token labels predicted by the CRF models and creating an enumerable representation of edges and nodes. The first method involved defining a formal context-free grammar (CFG) and building a deterministic parser for it; our second method was using a collection of heuristic strategies to simply pick these entities and relations out.

We invested most of our energy in constructing a CFG and tweaking a parser for that CFG, but also created the second, heuristic, interpreter for backup – the intuition being that a heuristic interpreter that was more conservative in extracting nodes/entities may result in higher precision scores, albeit at the expensive of reduced recall.

5.1 Defining a context-free grammar

We used a context-free grammar – assembled by manually inspecting the label sequences that underlie claims that we wanted to extract from the "remarks" text – in order to generate parse trees from sequences. The CFG has been modified to support token matching with *zero or more* [*] and *zero or one* [?] times. We also avoided left recursion to prevent unwanted parsing result.

All production rules for the CFG are reported in Appendix A of this document; each set of CFG rules is created to parse tokens falling with a particular "statement theme" tag.

Table 2: Context free grammar parsing example

Token	Tag
`"2"`	[t:person:NUMBER]
`"Ch"`	[t:person:CHILDREN]
`","`	[t:DELIMITER]
`"Jas"`	[t:person:NAME]
`"&"`	[t:DELIMITER]
`"Bridget Doyle"`	[t:person:NAME]
`","`	[t:DELIMITER]

5.2 Generating parse trees

Sequences of tokens and CRF-predicted labels are parsed using our context-free grammar in a deterministic manner. The choice to use a deterministic parser, as opposed to a method that learns to parse based on labeled training data, is rational because the types of claims which appear in the ESB "remarks" field are rarely, if ever, syntactically ambiguous. We made this decision after carefully surveying many examples of "remarks" entries, and the process of manually creating labeled training data for CRF purposes further enforced this decision.

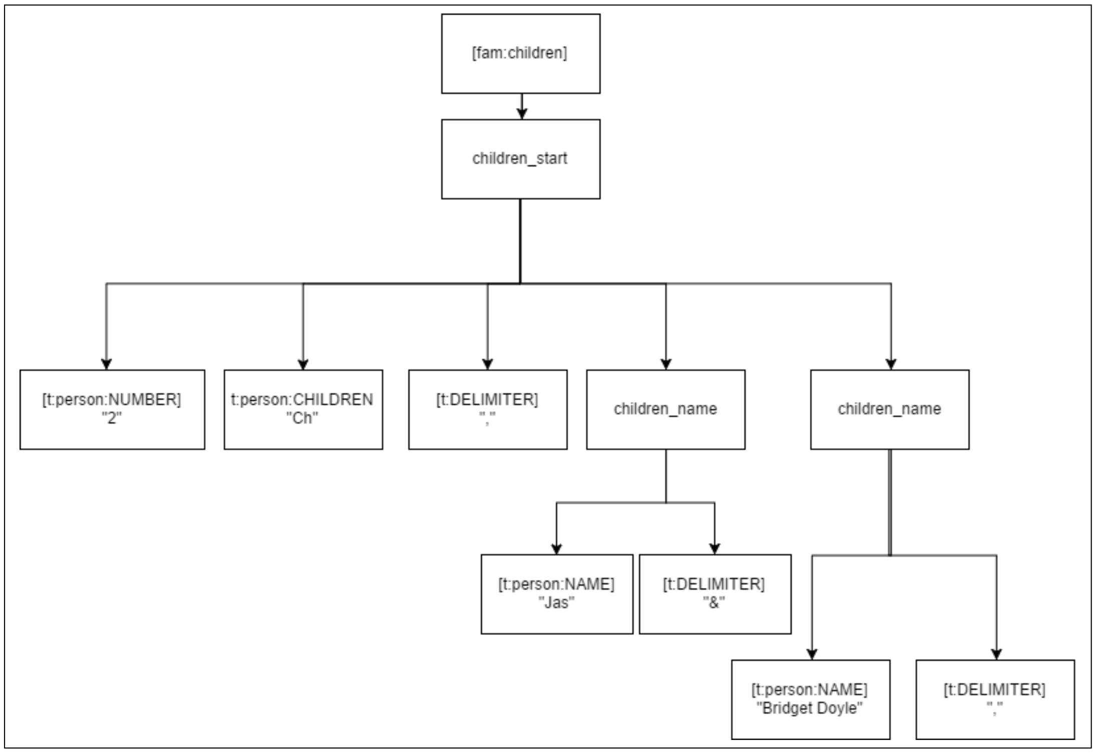


Figure 1: Example parse tree from the context-free grammar

Theme-specific CFGs are applied in a right-to-left manner on token sequences. With no left recursion, we reduced the number of passes to match an entire list of tokens with grammar rules. Figure 1 depicts a parse tree for a `fam:children`-labeled sequence of tokens and their labels.

5.3 Converting parse trees into discrete records

Our conversion algorithm for interpreting context-free grammar parse trees used a combination of breadth-first search (BFS) and depth-first search (DFS) algorithms for graph traversal. We used BFS when we traversed every root of a subtree (e.g. children_start). When we traversed any named entity subtree (e.g. children_name), we switched to DFS to parse property variables (e.g. children_location) and apply them to the named entity (e.g. t:person:NAME).

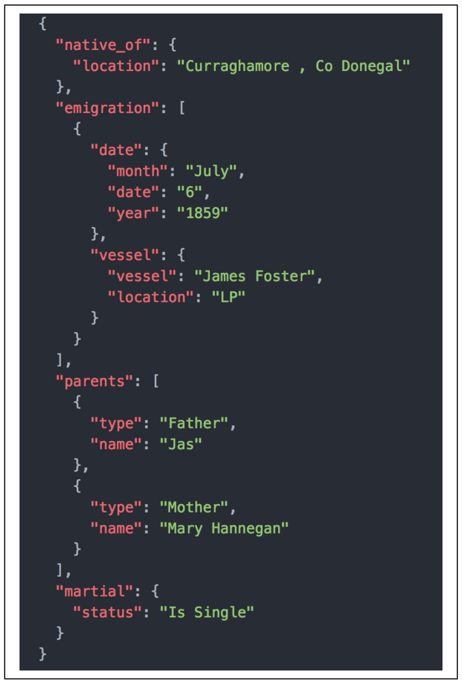
Using this technique, we were able to create records that clearly enumerate the nodes, edges, and properties that we want to represent in a graph network. The precision/recall of claims made in these structured-data records is also the evaluation metric that best represents the success of our system (as discussed later, in Section 7).

See Figure 2 for an example of an extracted record.

5.4 Heuristic sequence interpreter

As an alternative to the CFG-based approach, we started to implement a less formal interpreter of CRF-predicted label sequences that emphasized high-precision claims at the expense of lower recall. Fulling parsing a sequence of siblings, their attributes and locations, and names is something that is best done with a formal parsing mechanism, given the tree-like nature of this type of combination of descriptive modifiers with named entities; on the other hand, a heuristic interpreter might be able to more reliably list only the names of the siblings simply by searching for label sequences within a small n-gram window.

While more work could have been done to make the heuristic interpreter more complex, we decided to focus primarily on the CFG-based parser and interpreter. End-to-end scores comparing the performance of both are reported in the Evaluation section.



```
{
  "native_of": {
    "location": "Curraghamore , Co Donegal"
  },
  "emigration": [
    {
      "date": {
        "month": "July",
        "date": "6",
        "year": "1859"
      },
      "vessel": {
        "vessel": "James Foster",
        "location": "LP"
      }
    }
  ],
  "parents": [
    {
      "type": "Father",
      "name": "Jas"
    },
    {
      "type": "Mother",
      "name": "Mary Hannegan"
    }
  ],
  "martial": {
    "status": "Is Single"
  }
}
```

Figure 2: Example of an "extracted" record, from which graph nodes/edges can easily be read

5.5 Named entity normalization

Locations A cursory inspection of the ESB data reveals the necessity to perform named-entity recognition for placenames. Names of locations are frequently, but not consistently, abbreviated (e.g. "LP" for "Liverpool, England"). Even for non-abbreviated names, there is often such a degree of variance in styling or spelling that consolidating graph nodes for locations based purely on string matching would likely not be of much use.

We investigated methods for mapping location names onto authoritative placename identifiers, and settled on using data from the "Who's On First" gazetteer project, run by Mapzen.[3] This particular project synthesizes listings from a number of other placename authorities, including GeoNames.org, and issues identifiers than can be tied back to matching ids from these other sources of record. Additionally, Mapzen has developed an open-source geocoding schema on top of Elasticsearch, which we decided to use for the process of finding the best-matching authoritative entity given a location string.[4] Documentation for setting up a cluster of geocoding services, using Docker, and importing gazzetter data is available. (7)

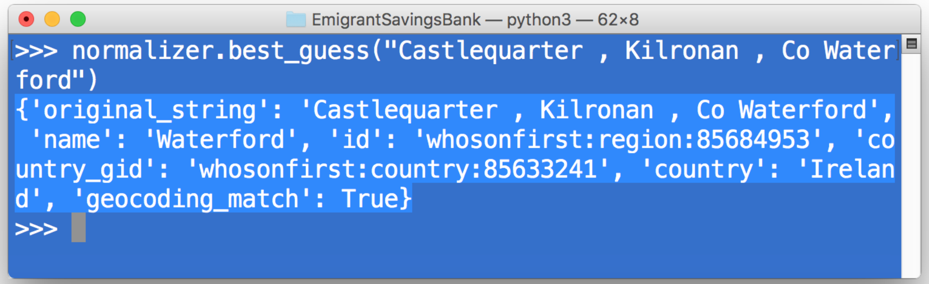
After assembling records with semantic labels and relations extracted for them, we ran all location strings through a normalizer that performed first some a priori substitutions (e.g., known domain-specific substitutions for this corpus, like "LP" into "Liverpool"), and then sent the modified string to our Pelias geocoding server to try to find a match. Geocoding is a messy process, and evaluation is difficult without significant historical and domain-specific knowledge. Therefore, our strategy was to pick the top matching authoritative ID (when any matches are found), but to retain the original string in the representation of the account node. Figure 3 depicts a sample invocation of the normalizer on a location string.

5.6 Constructing a graph

There seem to be many graph database implementations that are released as open-source projects, and any would probably suffice for our use. We decided to use Neo4j free/community edition, since it appeared to be full featured, it uses a query syntax that was not too burdensome to learn, and because

[3] `https://whosonfirst.mapzen.com/`

[4] Mapzen's Elasticsearch-based geocoder is called "Pelias": `https://github.com/pelias/pelias`



```
>>> normalizer.best_guess("Castlequarter , Kilronan , Co Water
ford")
{'original_string': 'Castlequarter , Kilronan , Co Waterford',
 'name': 'Waterford', 'id': 'whosonfirst:region:85684953', 'co
untry_gid': 'whosonfirst:country:85633241', 'country': 'Irelan
d', 'geocoding_match': True}
>>> 
```

Figure 3: Location normalizer interface in Python

it comes with an associated interface for visualizing graph queries. [5] Figure 4 depicts a subgraph in our Neo4j database.

6 Data model

In the process of creating token labels for the CRF models, the construction of context-free grammars, and the parsing of label sequences into itemized records, we had to make decisions about what types of entities and relationships between entities we wanted to support.

Many choices quickly become clear after reading 50 or so "remarks" entries. The enumeration of entities and relationships that can be found in our "extracted" records fits within the domain of entities and relations listed on Table 3.

Table 3: Relations (edges) and associated entities (nodes) in our graph data model

Relation types		
Entity	Relation	Entity
`person`	**is_native_of**	`location`
`person`	**is_located_at**	`location`
`person`	**has_father**	`person`
`person`	**has_mother**	`person`
`person`	**has_child**	`person`
`person`	**is_sister_of**	`person`
`person`	**is_brother_of**	`person`
`person`	**works_for**	`employer`
`person`	**lives_with**	`person`
`location`	**is_part_of**	`location`
`location`	**is_near**	`location`
`person`	**emigrated_via**	`voyage`
`vessel`	**made_voyage**	`voyage`
`voyage`	**departed_from**	`location`

6.1 Graph structure

Using a finite set of entity and relationship classes allows us to build a knowledge graph from parsed instances of these classes. The network depicted in Figure 4 shows an representative sub-graph of entities and relationships extracted from a single account's "remarks" section.

[5] `https://neo4j.com/`

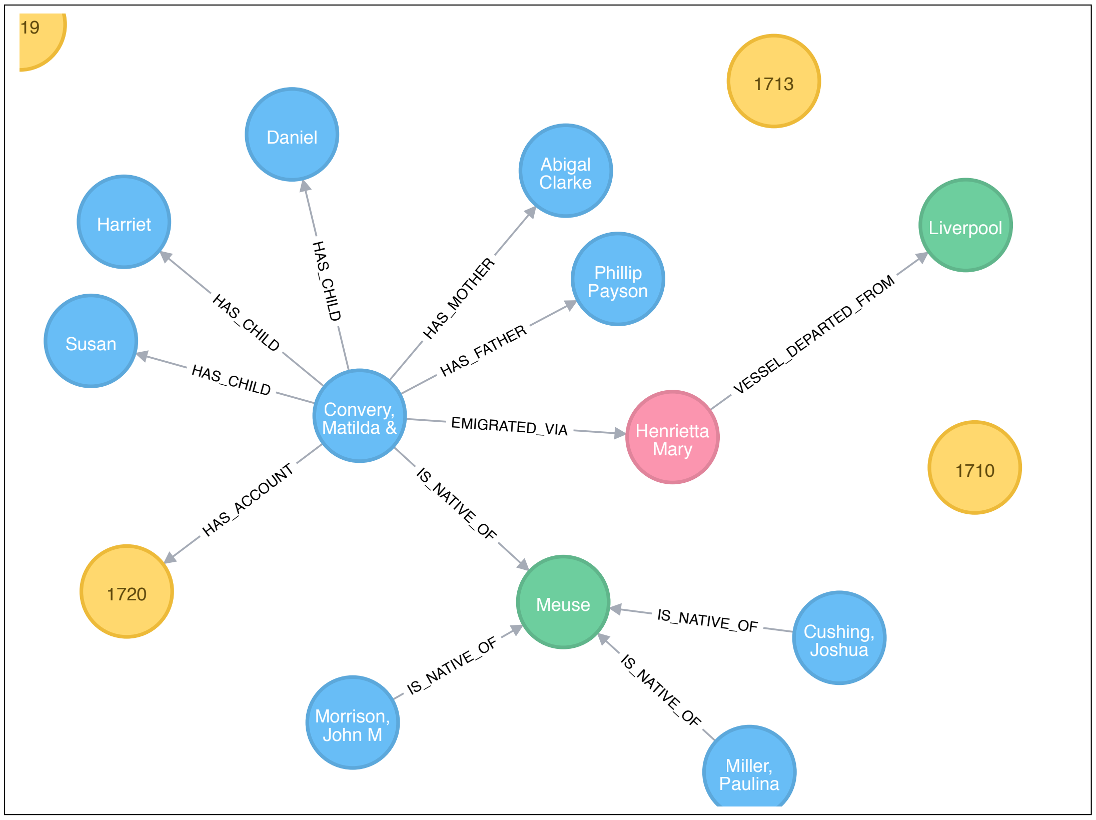


Figure 4: An example of a subgraph from extracted data (many edges not visualized); bank accounts are in yellow, individuals in blue, ships in pink, and locations in green.

Our hope is that by providing a reliable method for enumerating nodes and edges corresponding to the rich descriptions of entities and relationships within the natural language "remarks" of ESB accounts, historians with an interest in the data will have a far better ability to create sophisticated queries of the corpus.

7 Evaluation metrics

There are many stages in our pipeline at which accuracy could, or should, be measured: namely, both of the passes of CRF, the construction of parse trees, the processing or interpretation of parse trees, and finally at the normalization / authority-control stage during the construction of our graph.

We argue that the most relevant and meaningful accuracy metrics for our system are:

1. Accuracy of CRF labels from randomly sampled records
2. Precision / recall scores of represented "claims" in our parsed + information extracted versions of the records, also taken from a random sample

7.1 CRF accuracy

Since all labels used in the CRF process were created by the authors, and the training data was assembled by hand, we do not have enough labeled examples to use for evaluation in the form of a dev or held-out set. However, it is still possible to evaluate the accuracy of predicted labels by taking a random sample of original records that did not appear in the training set, running them through our two rounds of CRF, and manually counting the instances of mislabeling.

Out of the 30 records surveyed at random, the token-level labeler achieved an accuracy of `0.9694`, and the sequence-theme labeler never erred.

See Table 4 for complete results.

Table 4: Accuracy of CRF labels

CRF Pass	# Records	# Correct Labels	Incorrect Labels	Label Space Size	Accuracy
1 (theme)	30	1245	0	17	1.0
2 (token-level)	30	1207	38	66	**0.9694**

7.2 Precision / recall of claims in extracted records

We consider the most informative metric about the "end-to-end" performance of the system to be a set of precision and accuracy scores calculated on a random subset of 30 CRF-labeled, parsed, and extracted records.

We evaluate the system performance by considering every piece of structured information (i.e., each extracted node and relation between nodes) in terms of our established set of potentially extractable information. For this random evaluation, we considered only records with a "remarks" field that contained non-trivial remarks (therefore, records with a value of "No Remarks" were excluded).

For these metrics, we tally individual claims that are represented as nodes / edges in our extracted record, and compare those counts to the "gold" set claims that a human may extract from the corresponding original "remarks" text. One single claim could be that an individual *has 4 brothers*, while an additional four claims would be the modeling of four individual names of those brothers (i.e., "brother" nodes with names John, Pat'k, Martin, Nicholas).

Precision and recall are useful metrics for the type of task we are engaged in, since ours is fundamentally a problem of information extraction. These metrics allow us to characterize the difference between an interpreter that correctly identifies that a "remarks" section mentions, for example, two brothers along with their name, versus another interpreter that also identifies likewise but also (accurately) captures the fact that one of the brothers is currently located in the US.

Using our method of tallying discrete claims, we computed precision, recall, and a F_1 score for both the CFG-based and heuristic sequence extractors.

Full results are reported in Table 5.

Table 5: Claim modeling precision / recall metrics

Parser	# Records	# Gold Claims	Precision	Recall	F_1 Score
CFG	25	357	**0.9612**	**0.7647**	**0.8517**
Heuristic	25	357	0.9424	0.3669	0.5282

8 Discussion

In this project, we wanted to create a reliable method for converting the rich natural language statements from Emigrant Savings Bank account records into a graph of claims about people, locations, emigration voyages, and so forth. We believe that we have done this, and that our approach which combines a linear-chain CRF sequence model with a CFG parsing strategy is a viable means of extracting this type of data. Our intent is to now work together with Dr. Wolf on finalizing aspects of the data, and hopefully to make publicly available an extracted and graph-ready version of the entire corpus.

Some topics we wish to look into further:

1. Confidence metrics; knowing when a parse is less confident, as a signal to users of the data that they should manually investigate the claims made for a particular record
2. Modifications to our CFG that make it more robust to types of statements (in the form of label sequences) that we had not seen when creating the original production rules

3. Whether the use of more specific labels for the CRF stage might mitigate some problems at the CFG parsing end of things

9 Overview of attached code

We wrote approximately 2500 lines of Python code for this project. Some notes:

1. See the `README.md` file for annotated examples that show off some of the project API
2. The original 25k ESB dataset is at `data/esb25k.csv`
3. Our annotated training data is at `data/labels-training/esb_training_full.csv`
4. All of the attached code was written by us, however we call objects from the `sklearn-crfsuite` library [6]

Acknowledgments

The dataset was initially provided to the authors by Nicholas Wolf at New York University. The text transcription from microfilm copies of the original handwritten records was performed by Kevin Rich, and published by John T. Ridge.

References

[1] Chiticariu, L., Li, Y., & Reiss, F. R. 2013. "Rule-Based Information Extraction is Dead! Long Live Rule-Based Information Extraction Systems!". *EMNLP* 2013, pp. 827– 832. `https://aclweb.org/anthology/D/D13/D13-1079.pdf`.

[2] Dinu-Ionita, Andrei & Yu, Cong. 2016. "Knowledge, Web Search Engines Lecture 12". Accessed November, 2017. `http://cs.nyu.edu/courses/fall16/CSCI-GA.2580-001/lecture12.pdf`.

[3] Greene, Erica. 2015. "Extracting Structured Data From Recipes Using Conditional Random Fields". The New York Times Open Blog. Accessed November, 2017. `https://open.blogs.nytimes.com/2015/04/09/extracting-structured-data-from-recipes-using-conditional-random-fields/`.

[4] Jurafsky, Daniel & Martin, James H. Draft of August 7, 2017. "Information Extraction". In *Speech and Language Processing*. `http://web.stanford.edu/~jurafsky/slp3/21.pdf`.

[5] Lafferty, John, Andrew McCallum, and Fernando Pereira. 2001. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". Accessed November, 2017. `http://www.seas.upenn.edu/~strctlrn/bib/PDF/crf.pdf`.

[6] Salvato, Richard. 1997. "A User's Guide to the Emigrant Savings Bank Records". *New York Public Library*. Accessed November, 2017. `http://archives.nypl.org/uploads/documents/other_finding_aid/collection_1837_emigrant.pdf`.

[7] Shkolnikov, Diana. 2017. "Growing Pelias in Containers". Accessed December 16, 2017. `https://github.com/pelias/dockerfiles/blob/master/how_to_guide.pdf`

[8] Wolf, Nicholas. 2007. "Unpacking the Family Networks of the Emigrant Industrial Savings Bank Ledgers". Accessed November, 2017. `http://slides.com/nmarw49/esb-august-2017/`.

A Context-free grammars

The following pages reproduce the context-free grammar production rules used in the project.

[6] `https://sklearn-crfsuite.readthedocs.io/en/latest/`

Table 6: Context free grammar for `fam:parents` statements

LHS	RHS
`parent_start`	[parent_second] [parent_start*]
`parent_type`	[t:person:FATHER] [t:DELIMITER?]
	[t:person:MOTHER] [t:DELIMITER?]
	[t:person:PARENTS] [t:DELIMITER?]
`parent_location`	[t:person:LOCATED_IN] [t:location:NAME] [t:DELIMITER] [t:location:NAME] [t:DELIMITER?]
	[t:person:LOCATED_IN] [t:location:NAME] [t:DELIMITER?]
	[t:residential:CURRENTLY_LIVING_AT] [t:location:NAME] [t:DELIMITER?]
	[t:residential:CURRENTLY_LIVING_AT] [parent_location*]
	[t:residential:FORMERLY_LIVING_AT] [t:location:NAME] [t:DELIMITER?]
	[t:residential:FORMERLY_LIVING_AT] [parent_location*]
`parent_status`	[t:person:IS_DEAD] [t:DELIMITER?]
	[t:person:IS_LIVING] [t:DELIMITER?]
`parent_second`	[parent_type] [parent_status] [t:person:NAME] [t:DELIMITER?] [t:person:NAME*]
	[parent_type] [parent_location] [t:person:NAME] [t:DELIMITER?] [t:person:NAME*]
	[parent_type] [t:person:NAME] [t:DELIMITER?] [t:person:NAME*]
	[parent_type] [parent_location] [parent_status] [t:person:NAME] [t:DELIMITER?] [t:person:NAME?]
	[parent_type] [parent_location] [t:person:NAME]

Table 7: Context free grammar for `subj:emigration-event` statements

LHS	RHS
`emigration_start`	[*t:emigration:ARRIVED*] [*t:location:NAME*?] [emigration_date?] [*t:emigration:VIA*] [emigration_vessel]
`emigration_date`	[*t:time:MONTH*] [*t:time:DATE*] [*t:DELIMITER*?] [*t:time:YEAR*] [*t:DELIMITER*?]
	[*t:time:MONTH*] [*t:time:YEAR*] [*t:DELIMITER*?]
`emigration_vessel`	[*t:emigration:VESSEL*] [*t:emigration:VESSEL_HAS_ORIGIN*?] [*t:location:NAME*]

Table 8: Context free grammar for `fam:children` statements

LHS	RHS
`children_start`	[*t:person:NUMBER*] [*t:person:CHILDREN*] [*t:DELIMITER*?] [children_name*]
	[*t:subj:IS_MAN*] [*t:person:SON*] [children_name*]
`children_name`	[*t:person:NAME*] [children_location?] [*t:DELIMITER*?]
`children_location`	[*t:rel:IS_NATIVE_OF*] [*t:location:NAME*]

Table 9: Context free grammar for `fam:spouse` statements

LHS	RHS
`spouse_start`	[spouse_relation] [*t:DELIMITER*?] [*t:UNKNOWN**] [spouse_person*] [spouse_start*]
`spouse_person`	[*t:person:NAME*] [*t:person:IS_DEAD*?] [spouse_duration?] [spouse_location?] [*t:DELIMITER*?]
`spouse_relation`	[*t:rel:HAS_HUSBAND*]
	[*t:rel:HAS_HUSBAND*]
	[*t:rel:HAS_WIFE*]
	[*t:rel:HAS_SPOUSE*]
	[*t:rel:IS_WIDOW_OF*]
`spouse_location`	[*t:person:LOCATED_IN*] [*t:location:NAME*]
`spouse_duration`	[*t:time:YEAR*] [*t:time:DURATION_YEAR*]

Table 10: Context free grammar for `meta:record-reference` statements

LHS	RHS
`ref_start`	[*t:meta:SEE*] [ref_account*] [ref_start*]
	[*t:meta:IS_SAME_AS*] [ref_account*] [ref_start*]
`ref_account`	[*t:meta:ACCOUNT_NUMBER*] [*t:DELIMITER*?]

Table 11: Context free grammar for `subj:age` statements

LHS	RHS
`age_start`	[*t:person:AGE_YEAR*] [*t:person:AGE*]

Table 12: Context free grammar for `subj:biographical-note` statements

LHS	RHS
`bio_start`	[*t:time:DURATION_VALUE*] [*t:time:DURATION_YEAR*] [*t:person:LOCATED_IN*] [*t:location:NAME*

Table 13: Context free grammar for `subj:marital-status` statements

LHS	RHS
`marital_start`	[*t:person:IS_SINGLE*]

Table 14: Context free grammar for `subj:native-of` statements

LHS	RHS
`native_of_start`	[*t:rel:IS_NATIVE_OF*] [*t:location:NAME*] [*t:DELIMITER*?] [native_of_dist?] [native_of_start*]
`native_of_dist`	[*t:location:DISTANCE*] [*t:location:DISTANCE_UNIT*?] [*t:location:FROM*] [*t:location:NAME*]

Table 15: Context free grammar for `subj:occupation` statements

LHS	RHS
`occupation_start`	[*t:work:WORKS_FOR*] [*t:work:EMPLOYER_NAME*] [*t:work:OCCUPATION*]

Table 16: Context free grammar for `subj:residence-info` statements

LHS	RHS
`residential_start`	[*t:residential:LIVES_WITH*] [residential_person*] [residential_start*]
	[*t:residential:LIVED_WITH*] [residential_person*] [residential_start*]
	[*t:residential:CURRENTLY_LIVING_AT*] [*t:DELIMITER*?] [residential_location] [residential_person*] [residential_start*]
	[*t:residential:FORMERLY_LOCATED_AT*] [*t:DELIMITER*?] [residential_location] [residential_person*] [residential_start*]
`residential_person`	[*t:person:NAME*] [*t:DELIMITER*?]
	[*t:person:NAME*] [*t:DELIMITER*?]
	[*t:person:SON*] [*t:DELIMITER*?]
	[*t:person:MOTHER*] [*t:DELIMITER*?]
	[*t:person:FATHER*] [*t:DELIMITER*?]
	[*t:person:BROTHERS*] [*t:DELIMITER*?]
	[*t:person:SISTERS*] [*t:DELIMITER*?]
	[*t:person:WIFE*] [*t:DELIMITER*?]
	[*t:person:CHILDREN*] [*t:DELIMITER*?]
`residential_location`	[*t:location:NAME*] [*t:DELIMITER*?]