Extracting Fields from Free-Text

by Pedro Cattori

S.B., C.S. MIT 2014

Submitted to the
Department of Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2016

Author: _____

Department of Electrical Engineering and Computer Science
January 29, 2016

Certified by: _____

Samuel Madden
Thesis Advisor

Accepted by: _____

Christopher J. Terman
Chairman, Master of Engineering Thesis Committee

Extracting Fields from Unstructured Text
by Pedro Cattori
Submitted to the Department of Electrical Engineering and Computer Science

January 29, 2016
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Field Extraction Library (FEL) provides functions for named-entity extraction within free text. FEL models the content structure of the specified named-entities rather than relying on brittle, context-specific separator logic.

Users specify the names of the fields they wish to extract, which determine the number of states for an underlying Hidden Markov Model. The observable emission set is pre-determined by FEL's tokenizer. Once the model topology is set, users provide training examples of the form:

$$x = \text{raw text}, y = \{\text{field1: val1, field2:val2, } \ldots\}$$

FEL learns the parameters of the underlying Hidden Markov Model by maximum likelihood model-estimation on the training examples.

FEL is designed to operate on small, sparse training data. As a result, users can provide few (less than 10) training examples to bootstrap the model. FEL offers 3 iterative mechanisms for scaling data quality as users provide guidance through additional feedback: (1) accept more training examples, (2) create *landmark states*, and (3) bridge related states with *state bridges*. FEL detects ambiguities both in its internal model and in the extraction results to prompt users for more feedback.

Once the model yields acceptable result quality, users can extract fields into a table for easy querying and exporting.

Acknowledgments

First, I would like to thank my advisor Sam Madden for his support throughout this project. Sam trusted me to tackle the problem in my own way and connected me with all the people who made this possible.

I would also like to thank Anant Bhardwaj for his diligent explanations and for laying the groundwork of the Field Extraction Library in his Distill paper (unpublished).

I thank Paresh Malalur for taking the time to discuss Hidden Markov Models with me at length near the beginning of this endeavor.

Finally, I thank my mother Karol, my father Pedro, and my girlfriend, Isabella Hurtado-Braun, for their unflinching support when I was fully devoted to my research.

Contents

1. Introduction

Unstructured data is flexible; data does not need to fit into a structure like a table schema. Its versatility makes it a popular solution for storing manually-entered data such as physicians' notes or product descriptions. In these situations, free-text is often the chosen implementation of unstructured data.

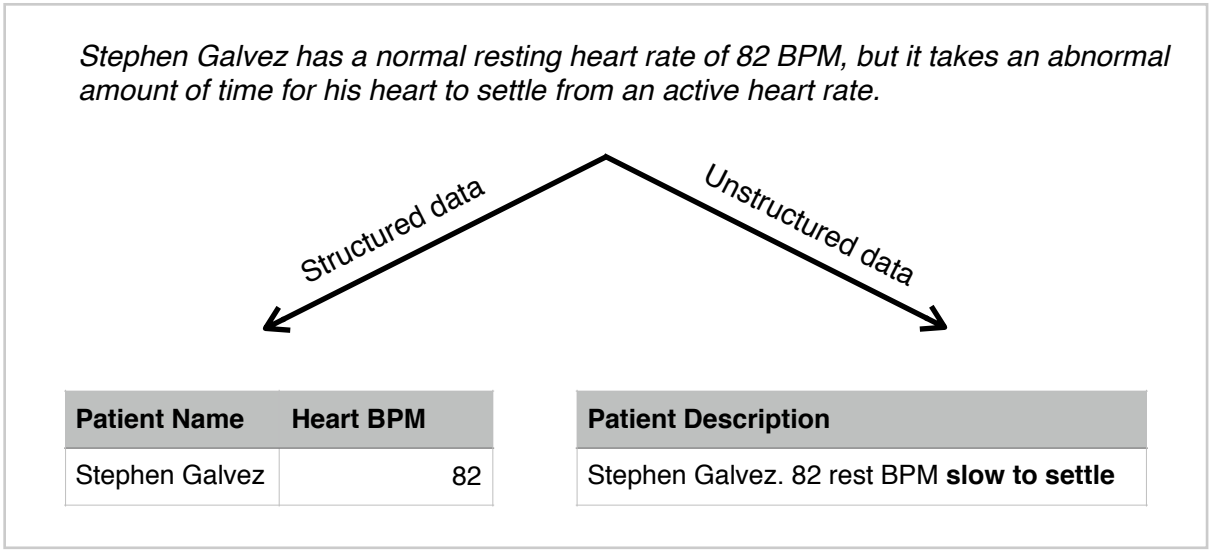


Figure 1: *Flexibility of free-text data allows physician to log abnormality in data system.*

Figure 1 highlights how information-loss can occur with structured data.

While inputs are extremely permissive for free-text, querying free-text is problematic. The user has 2 options: (1) parse the free-text or (2) accept poor query performance of slow string comparisons. **Section 1.1.1** examines why (1) parsing the free-text cannot be done by hand.

An ideal solution should take the best of both worlds: flexible free-text input with performant table queries.

In this paper, I describe my design for a Field Extraction Library (FEL) to implement the ideal flexible-yet-performant solution. The key insight is to *model the content structure* of the fields.

1.1 Problem Description

The goal of FTFE is to put together tabular views of free-text data.

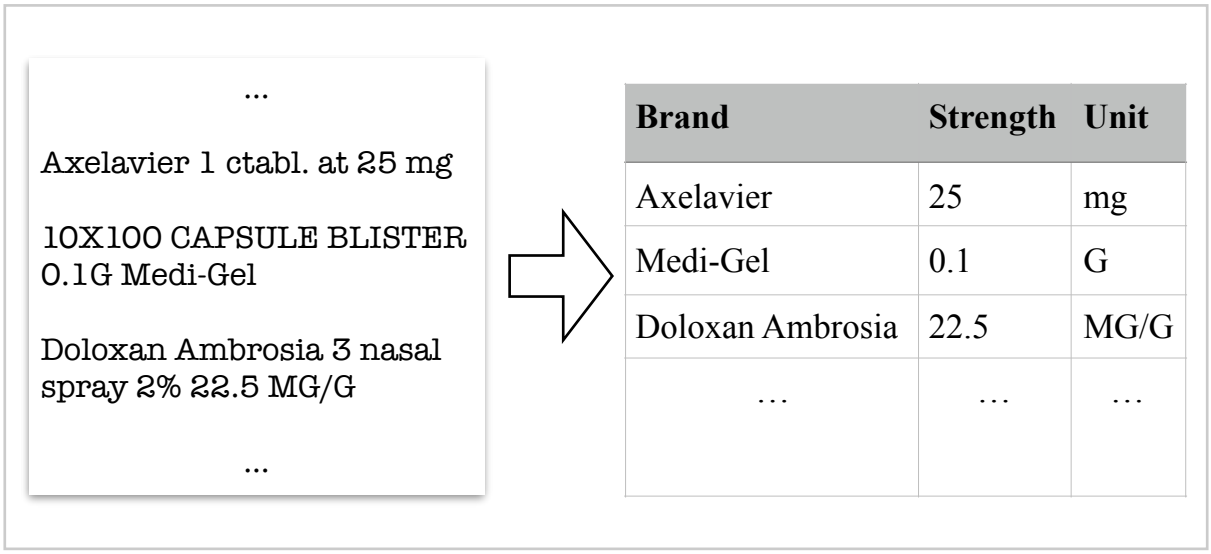


Figure 2 : *Free-Text Field Extraction applied to pharmaceutical product descriptions.*

Figure 2 shows how solving the Free-Text Field Extraction (FTFE) problem realizes the ideal solution by transforming free-text into a data table.

FTFE is applicable to many real-world datasets such as hospital records, catalogs with product descriptions, membership renewal services, etc. Frequently in situations, legacy systems with free-text data are already in place and ignoring the data in them is unacceptable. The appendix in **Section 7** contains example datasets for the domains enumerated here. Additionally, many publicly-available datasets, especially web-scraped data, were collected as free-text. Solving FTFE offers a scalable way to finally make all this data accessible.

FTFE is related to the Named-Entity Extraction (NER) problem described in **Figure 3** below.

Most research on NER systems has been structured as taking an unannotated block of text, such as this one:

Jim bought 300 shares of Acme Corp. in 2006.

And producing an annotated block of text that highlights the names of entities:

[Jim]$_{\text{Person}}$ bought 300 shares of [Acme Corp.]$_{\text{Organization}}$ in [2006]$_{\text{Time}}$.

Figure 3 : *Wikipedia's illustrative example of Named-Entity Recognition*

FTFE differs from NER in 2 key dimensions: (1) Unlike NER, FTFE operates on free-text, not languages with well-studied vocabularies and grammars. (2) Instead of annotating text, FTFE outputs data tables. **Section 1.1.1** explores the challenges of free-text as input. Consequences of tabular output are examined in **Section 1.1.2**.

1.1.1 Free-Text

Free-text is a digital representation of human note-taking and exhibits 2 key characteristics that make free-text extraction challenging.

(1) **Inconsistent** : A free-text dataset presumably contains individual records, each containing similar pieces of information. However, each record may contain extra information (noise), missing values. Across records, formatting is consistent.

(2) **No language model** : Free-text data often contains words from a human language (English for the figures in this paper). Yet, the data does not conform to any grammar models or part-of-speech ordering rules. Furthermore, preprogrammed language models like those used by Basis Tech (**Section 1.2.1**) cannot cope gracefully with the frequent typos in free-text.

These 2 challenges are addressed in **Section 1.1.3**.

Prescription Pick-Up
On 10/3/2009 benbitd@mit.edu (+1-470-555-8321) bought 64x Cordrazine for $8.79
tim.fritz@gmail Zydrate REFILL 60 TABLETS 7.19.2004 PAID $25.95 pre-tax
$37.99 Hourai X (20 pills) date:04.15.13 to Mr. Vittorio - jvittori@akamai.net 623-555-0843
5/22/14 - ehatzle@yahoocom pick-up 48 pack Alprazaline - $42.87 USD
15 Progenitorivox at $10.50 refilled for (james77@inbox.com, 8795552304) : 11/14/07

Figure 4 : *Prescription Refill dataset containing the 2 challenges of free-text. (1) Data is not created from a common template. Missing phone numbers for 2nd and 4th records. (2) Though clearly written in English, text does not follow normal language patterns like part-of-speech rules. Typos like the "." missing in the 4th record's email.*

1.1.2 Tabular Output

In practice, most data operations are performed on structured data, specifically in the form of database tables or other tabular representations (e.g. Excel spreadsheets, CSV files, etc…). Tables represent data in a form that is efficient for querying and can be linked to other data via relational techniques. To this end, my design provides its output as CSV file which can easily be ingested into a database or other tabular data management solution.

The constraint of tabular output gives rise to the challenge of record slicing.
Splitting free-text into pieces corresponding to records is called *record slicing*. Record slicing is assumed to be a solved problem for the purpose of this research, but possible solutions to the record slicing problem are laid out in **Section 5.1** of **Future Work**.

1.1.3 Field Extraction Library

The remaining sections of this thesis covers the Field Extraction Library (FEL) I developed and how it address the challenges laid out in **Section 1.1.1**. Solutions to the challenges from **Section 1.1.2** are covered in **Section 5.1**.

To address challenge (1), FEL cannot depend heavily on separator logic. Separator logic builds conditional rules on the formatting of the data. Instead, FEL introduces the concept of *content structure modeling.* For (2), the underlying Hidden Markov Model's state transitions acts as a simple, learned language model of the free-text.

Additionally. FEL is based on probabilistic approaches that gracefully handle extra data, missing data, and errors such as typos.

In FEL, preprogrammed language models, and rule-based approaches are not obsolete; in fact, FEL incorporates these techniques in its design. The important point here is that FEL is not overly reliant on any one of these techniques thanks to its principled, machine-learning foundations. **Section 1.2** discusses previous work, all of which I believe to be overly reliant on separator logic and preprogrammed language models.

1.2 Previous Work

Previous work can be categorized into 2 groups: (1) relies heavily on separator logic and (2) depends on preprogrammed language model. While some previous work exists to output tabular data, both approaches have intrinsic shortcomings:

(1) is provably brittle as it overly depends on consistency of conventions such as formatting. This approach makes no attempt to semantically understand the fields and can be thought of as automated building of **sed** and **awk** scripts.

(2) cannot gracefully handle text that does not conform to language rules. Any word omissions, typos, or improper grammar represent failures modes for this approach.

Neither of the 2 groups can gracefully handle non-linguistic and convention-inconsistent data.

1.3.1 Basis Tech

Basis Technology describe themselves as "the leading provider of software solutions for extracting meaningful intelligence from *multilingual* text and digital devices" (Basis 2013, italics mine).

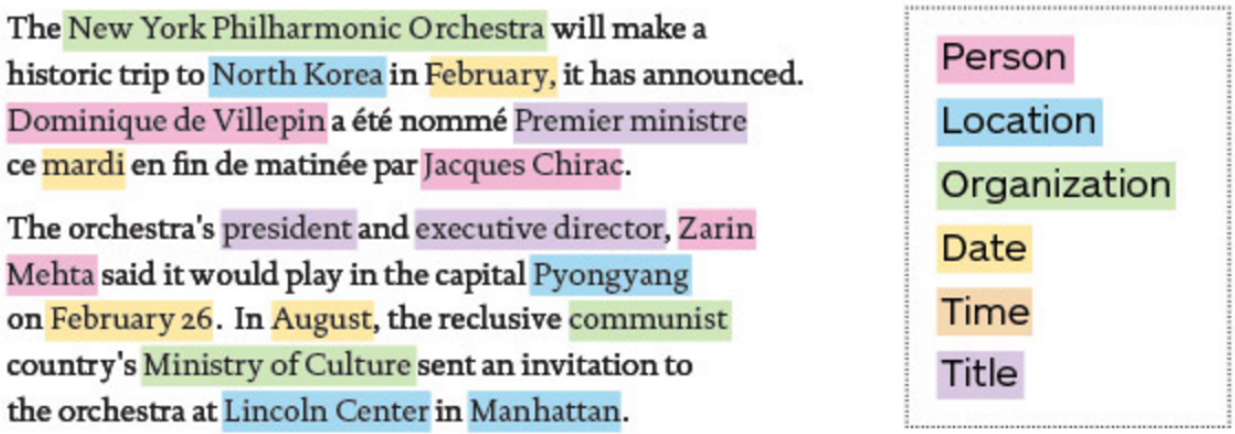


Figure 5 : *Basis Tech.'s Rosette Entity Extractor labeling an example dataset.*

Figure 5 demonstrates the capability of their aptly named Rosette Entity Extractor (REE) product. The functionality of REE heavily depends on built-in linguistic models of human languages. REE's competitive advantage leverages dictionaries of known words and part-of-speech rules along with a suite of carefully crafted custom rules on a per-language basis.

While REE's results are impressive for the examples shown, it was not designed to handle non-linguistic data. As a result, non-linguistic datasets result in poor result quality.

1.3.2 Flash-Extract

Flash-Extract started as research within Microsoft (Le et al. 2014). The goal is to extract entities from transactional, consistently-formatted data. Flash-Extract targets machine-generated text with low content variability. Its HCI is meant to build regular expression parsing for non-technical users.

Machine-generated text rarely has inconsistencies such as typos or missing data. Typically, machine-generated text is derived from a set of templates. Tempting enforces a consistent data layout. Thus extraction is accomplished by understanding the *surrounding data format* rather than understanding the content structure for each field.

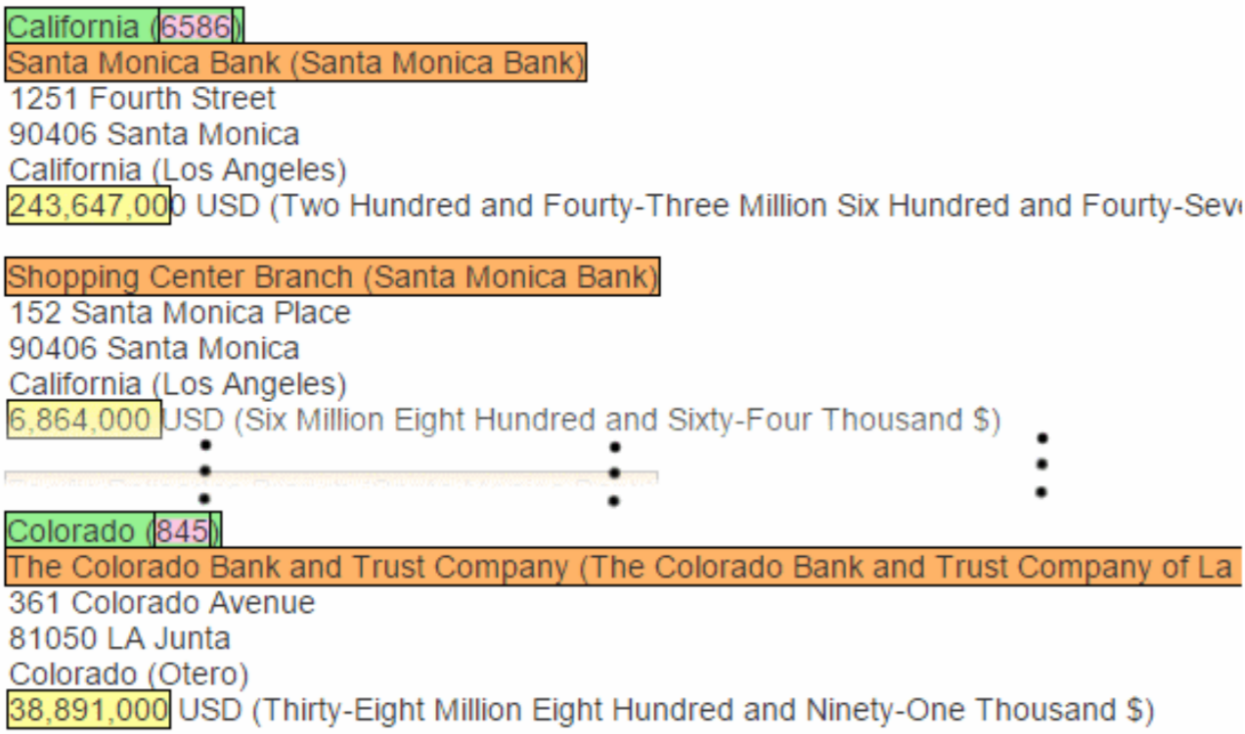


Figure 6 : *Flash-Extract example from their paper (Mayer et al. 2015)*

Figure 6 shows Flash-Extract recognizing fields in a low content variability setting. Notice that fields always appear in the same order and that fields can be found by describing their relative location rather than what they look like. For example, 'dollar amount' starts on the 6th line and

goes until the first space; notice how there is no semantic understanding about 'dollar amount' being a large number often followed by the domain-specific term 'USD'.

1.3.3 Data Wrangler

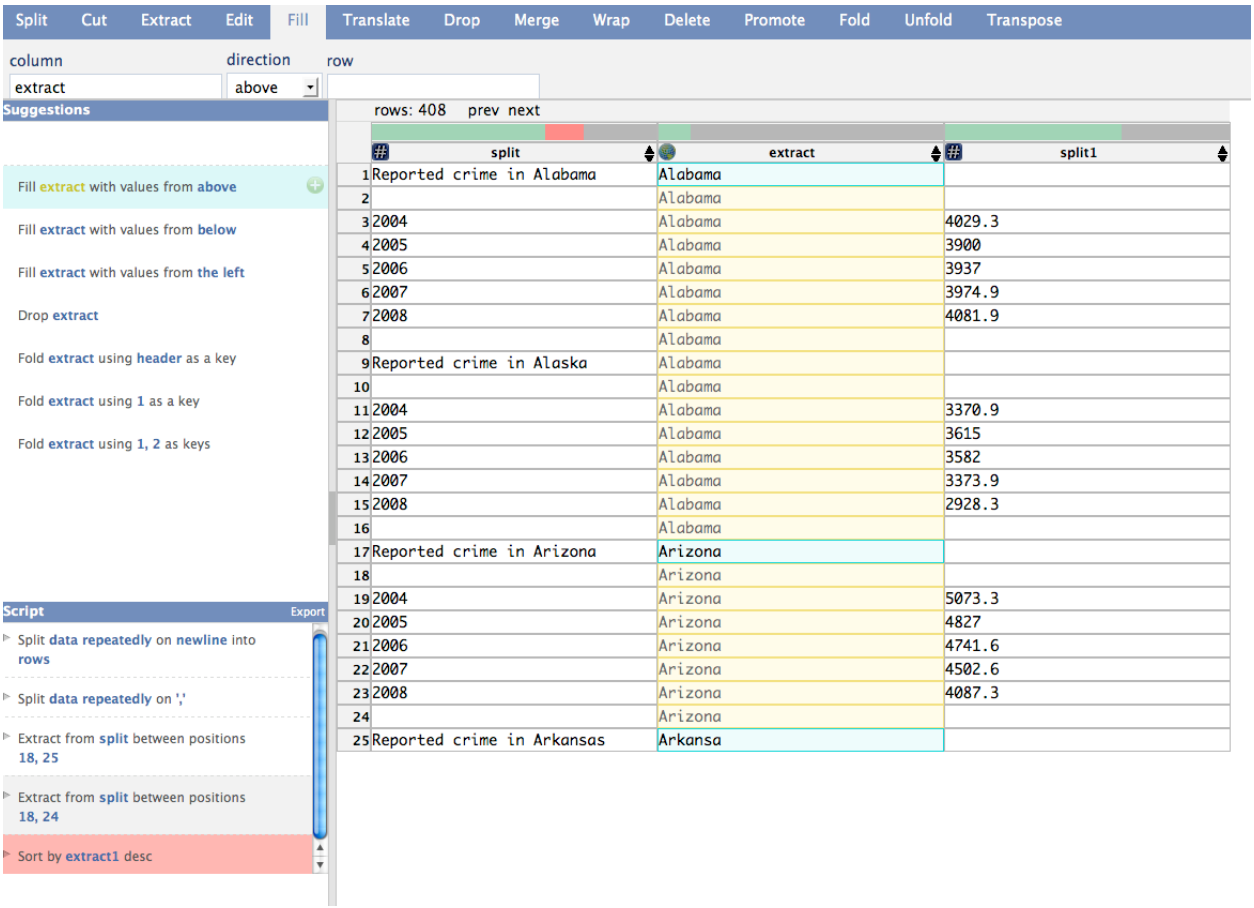


Figure 7 : *The Data Wrangler graphical user interface running on crime data*

Data Wrangler excels at transforming data by providing an interface for users to split, merge, sort, etc… their data (Kandel et al. 2011). These operations are typical of an Extract-Transform-Load (ETL) tool. As can be seen in the lower lefthand quadrant of **Figure 7**, Data Wrangler saves user operations as a script. Data Wrangler transformation scripts heavily rely on separator logic (eg. Extract from split between positions 18,24) instead of modeling content structure.

Like Flash-Extract, Data Wrangler targets machine-generated, low content-variability text data. And like Flash-Extract, its rule-based approach is brittle for cases where separator logic is insufficient.

1.4 Key operating principles (KOPs)

1.4.1 Optimized for Common-Case, Scale to Worst-Case (OCCSWC)

This design principle dictates that users should have a simple interface for simple use cases. In other words, default behavior should handle common problems and take care of low-hanging fruit. If users want more configuration or advanced features, they may have to use more complicated interfaces.

For FEL, this means that users who do the bare minimum can still arrive at results that extract easy-to-find fields, but can provide further input to incrementally reach better result quality.

1.4.2 Machine Driven, Human Guided

"All models are wrong, but some are useful" (Box 1978)

I embrace the Boxian view that models (and by association, machine learning) will not be able to perform perfectly on any real-world task. Luckily, human expertise can often cover the failure modes of machine learning. A tight iterative loop of human-machine collaboration (referenced by the moniker "Machine driven, human guided") takes the best of both worlds: the scalability of machine learning and the specific background knowledge of humans (Tamr 2015).

Human-machine cooperation necessitates direct human-computer interface (HCI). Specifically, the machine should only bother humans when necessary and only with 'bang for your buck' questions. Questions should be crafted to optimize information gain while minimizing human involvement.

1.4.3 Principled Learning

Rule-based approaches rely mostly on context-specific patterns. While a rule-based approach is powerful in its context, it cannot be easily generalized to different contexts. In other words, rule-based approaches tend to suffer from overfitting and are brittle in new contexts.

A great platform may exploit context-specific patterns via rule-based algorithms, but must also have a principled foundation to fall back on when it needs to generalize well across problem instances.

Rule-based systems are an achievement of engineering, not of modeling (Marr 1976). Though FEL is implemented with some hard-coded rules, **Section 5** outlines how these pieces can be swapped out in favor of more principled, learning-based implementations.

2. Representation

"Once a problem is described using an appropriate representation, the problem is almost solved." (Winston 1993)

A suitable representation must address the 2 challenges of free-text covered in **Section 1.1.1**: inconsistencies and lack of language model. To cope with inconsistencies, FEL cannot rely on rule-based logic, but rather embrace a probabilistic approach. To compensate for the lack of a preprogrammed language model, FEL must learn a language model for the data at hand.

By learning a probabilistic model, FEL is able to identify problematic cases (eg. extra data, missing data, typos) and handle them gracefully.

To learn a probabilistic model, FEL needs to employ machine learning.

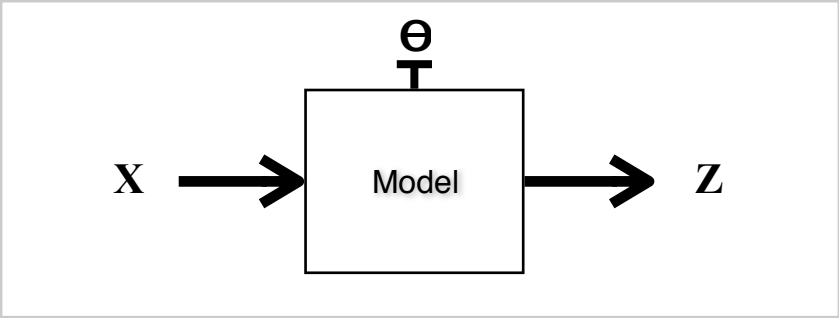


Figure 8 : *Machine learning produces a model that receives input, **X**, and outputs predictions, **Z**. The model can be tuned by adjusting its parameters,* **Θ**.

Analogously to **Figure 8**, FEL's input **X** refers to a semantic view of the free-text. **Section 2.1.1** discusses how FEL transforms the raw free-text it receives into the semantic units comprising **X**. The output **Z** corresponds to the most accurate extractions of the free-text **X** as ranked by FEL's model. **Z** must then be transformed from its semantic representation to a physical data table (eg.

CSV file). [refer to Caltech's learning from data book] **Section 2.1** and **Section 2.2** motivate the choice of Hidden Markov Models as FEL's underlying model.

Figure 9 exposes the models dependencies, namely the parameters **Θ**. In accordance to the Machine Driven, Human Guided KOP of **Section 1.3.2**, FEL cannot rely on humans to set its parameters directly as this would require the users to have deep understanding of the underlying model. Instead, FEL leverages supervised machine learning to automatically tune the parameters according to a small set of examples provided by the user.

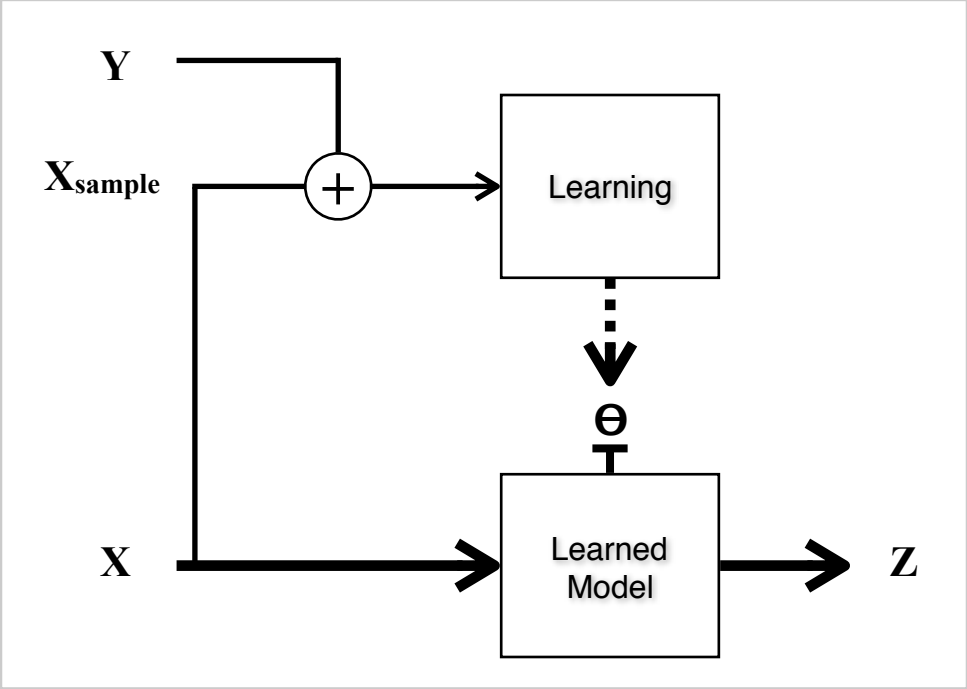


Figure 9 : *Supervised machine learning samples the dataset* ***X*** *to produce* $\boldsymbol{X}_{sample}$*. This sample is combined with labels provided by the users that specify the correct extractions for each record in* $\boldsymbol{X}_{sample}$*. A learning algorithm then automatically tunes the parameters* **Θ** *of the model. The + operator combines each individual free-text record* ***x*** *and corresponding extraction example* ***y*** *into a training pair* ***(x, y)****.*

Figure 9 shows how supervised machine learning removes the burden of parameterization from the user, enabling non-technical users to indirectly train the model via labeled examples. **Section 2.3** covers how FEL's model generates its predictions, **Z**, and explains FEL's learning algorithm.

Section 2.4 examines how FEL copes with noise and errors. **Section 2.5** discusses extensions to improve FEL's model.

2.1 Content-Structure Models

To address Challenge (1), FEL must depend on reliable patterns in text-free. Formatting is immediately ruled out as unreliable due to its inconsistency. Instead, FEL models the structure of the contents of a field. By doing so, FEL gains some semantic understanding of the fields; FEL will learn what fields "look like".

Prescription Pick-Up
On 10/3/2009 benbitd@mit.edu (+1-470-555-8321) bought 64x Cordrazine for $8.79
tim.fritz@gmail Zydrate REFILL 60 TABLETS 7.19.2004 PAID $25.95 pre-tax
$37.99 Hourai X (20 pills) date:04.15.13 to Mr. Vittorio - jvittori@akamai.net 623-555-0843
5/22/14 - ehatzle@yahoocom pick-up 48 pack Alprazaline - $42.87 USD
15 Progenitorivox at $10.50 refilled for (james77@inbox.com, 8795552304) : 11/14/07

Figure 10 : *Free-text prescription data with inconsistent formatting. Emails, dates, brands, etc… have no relative ordering. 1st and 5th records use parentheses, 3rd record uses colons, 4th record uses hyphens. 2nd and 4th records are missing phone numbers.*

Separator logic is too crude to recognize phone numbers in **Figure 10**. Yet, humans have no trouble identifying *1-470-555-8321*, *623-555-0843*, and *8795552304* as phone numbers. The key insight is that these strings *look like* phone numbers because of their structure, not the surrounding data.

Data	Phone number?	If not, why?
4845553752	Y	
7xwo81jfg	N	letters
239@085#7292	N	weird symbols
+1-470-555-8321	Y	
958-3332	Y	
- - 18472 2341 3 - -	N	out of order
213.555.8344	Y	

Figure 11 : *Phone numbers can be identified even when they are taken out of context.*

Notice that the reasons provided in **Figure 11** are comments on the *structure of the content*.

Section 2.1.1 examines tokenization to represent data as semantic chunks, so the FEL can reason about groups of digits, words, and punctuation instead of character-by-character analysis.

Section 2.1.2 introduces the probability mass function (PMF) model as a simple context-structure model and discusses its limitations. One limitation of the PMF model is that it will incorrectly identity "*- - 18472 2341 3 - -*" as a phone number since it only models token-membership, not token ordering. More advanced content-structure models that capture internal token ordering are discussed in **Section 2.5** and **Section 5.3**.

2.1.1 Semantic Chunks

The first step is to take the physical input and convert it to semantic chunks called *tokens*. These basic semantic units are essential to model-making (Trim 2013).

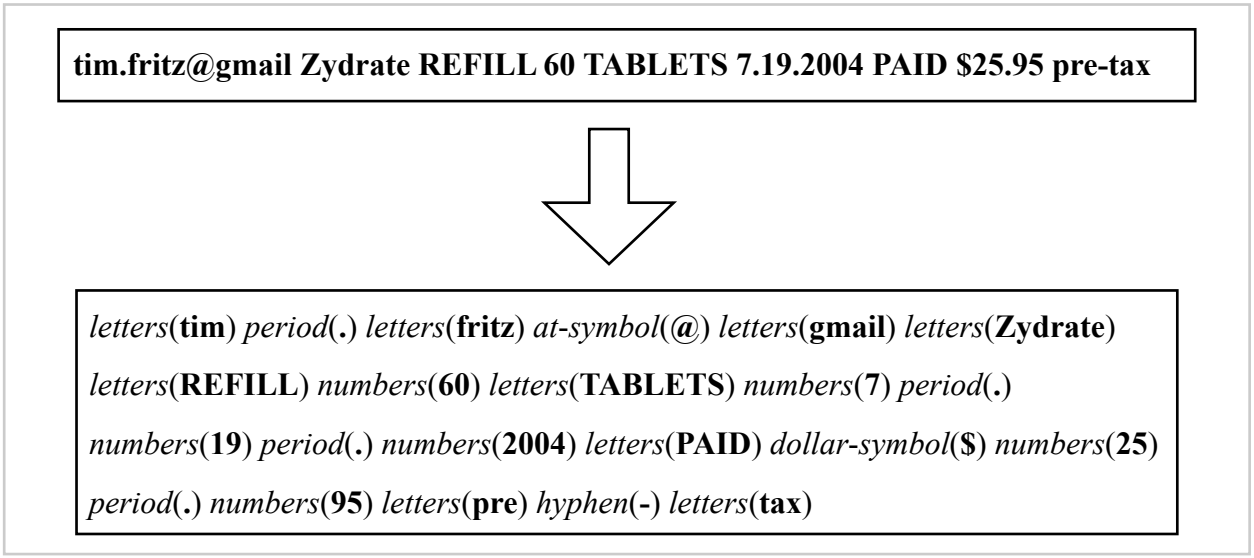


Figure 12 : *Tokenization with* **letters**, **numbers**, **at-symbol**, **dollar-symbol**,*and* **hyphen** *as token types (whitespace omitted).*

Figure 12 demonstrates how tokenization transforms bytes (visualized as raw text) into semantic units. Tokenization allows FEL to base its model on syntactic tokens rather than meaningless characters. For FTFE, feature selection occurs during this tokenization step.

2.1.2 PMF Fields

As **Figure 13** demonstrates, probability mass functions (PMF) treat fields as "bags of token types", ignoring any notion of ordering between token types.

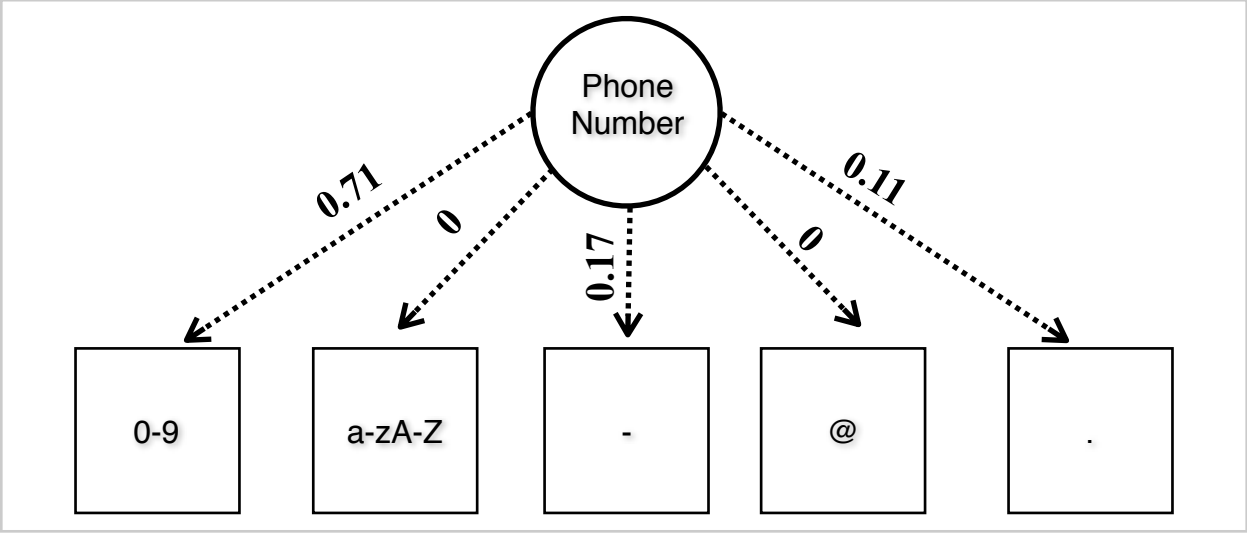


Figure 13 : *Phone number PMF over 5 token types (digits, letters, hyphens, at-symbol, and period) represented as a graph. The graph models phone numbers as made up of many digits, some hyphens, and fewer periods. PMF probabilities must sum to 1.*

While clearly oversimplified, this model is able to identify all data correctly in **Figure 11** except for the *out-of-order* case. **Section 2.5** applies the concepts covered in **Section 2.3** to model with token-ordering within a field.

Frequently, the sparse training data may not expose certain token type combinations for a field. To cope with the sparseness of the training data, FEL employs smoothing. For a more detailed account of smoothing across my design, refer to **Section 3.5**.

2.2 Learning Grammar

While free-text does not comply to human language rules, it stills exhibit linguistic properties. **Figure 14** shows what a simple language model might look like for free-text data.

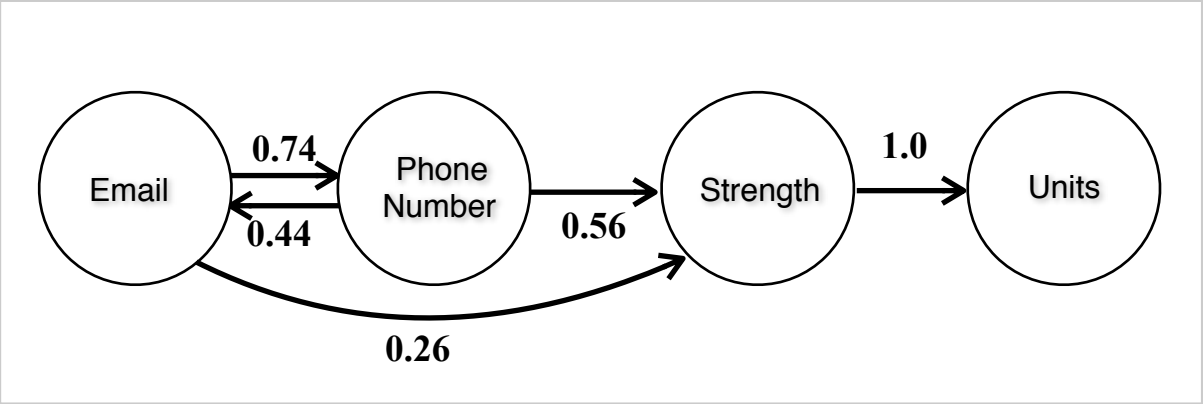


Figure 14 : *One possible part-of-speech grammar model for email, phone number, strength, and unit fields in prescription data. Number labels represent the transition probabilities.*

FEL is designed to *learn* language models like this one, removing the burden of understanding free-text linguistically from the user.

2.3 The Hidden Markov Model

Figure 15 combines the diagrams in **Section 2.1** and **Section 2.2** into one representation capable of modeling content-structure and grammatical part-of-speech ordering simultaneously.

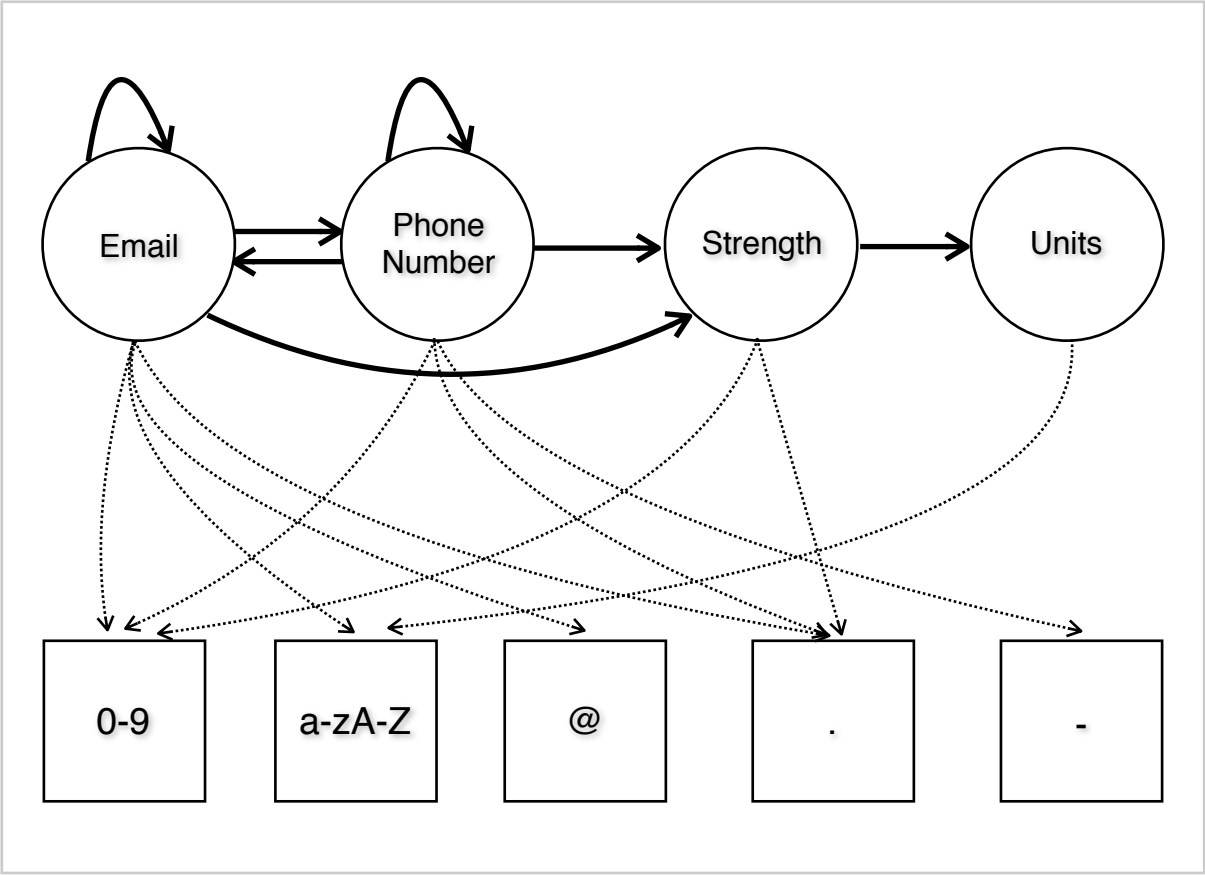


Figure 15 : *Content-structure modeling for each field across a set of common token types (model drawn with dotted arrows) and part-of-speech model between fields (model drawn with bold arrows). Probability labels omitted to avoid cluttering the diagram.*

Figure 15 exposes the Hidden Markov Model (HMM) as the inherent representation for FEL's model; fields are the states of the HMM and token types are the observables emissions. Interpreting FEL's model as an HMM, transitions between fields now correspond to moving between adjacent tokens. Since two adjacent tokens correspond to the same state, self-loops have been added to the appropriate fields in **Figure 15**. Thus self-loops correspond to transitioning

between tokens in the same field whereas other state transitions correspond to switching between fields.

Hidden Markov Models are characterized by their topology and parameters as encoded in their transition probability matrix A, emission probability matrix B, and initial state probability vector π (Rabiner 1990). Specifically, the topology of the HMM determines the dimensionality of A, B, and π while the parameters specify the values contained in each. For more background information on HMMs refer to (Rabiner 1990)'s *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*.

Three main computational problems for HMMs

The following three fundamental problems are the building blocks of HMMs:

- **P1 (Evaluation)**. Compute the likelihood $P(X|\Theta)$
- **P2 (State-estimation)**. Find "best" state-sequence; $\max_Y \ P(Y|X,\Theta)$
- **P3 (Model-estimation)**. MLE for parameters; $\max_\Theta P(X|\Theta)$

Figure 16 : *Hidden Markov Model lecture notes from MIT's 6.036 Machine Learning class. Notation is different.*

Figure 16 defines State-estimation and Model-estimation as computational problems solved by HMMs. State-estimation is an out-of-the-box HMM inference solution for FTFE, but depends on a tuned HMM. Model-estimation ensure that the learned model required for state-estimation is tuned. **Section 2.3.1** covers state-estimation in the context of FTFE and **Section 2.3.2** does the same for model-estimation.

2.3.1 State-estimation

The goal of state-estimation is to determine the most likely sequence of states **z** for a set of observations **x**.

$$z_1^*, ..., z_T^* = arg \max_{z_1,...,z_T} P(z_1, ..., z_T, x_1, ..., x_T)$$

Figure 17 : z^* *is the predicted state sequence that maximizes probability according to the HMM.* ***x*** *is sequence of observations. The equation expands both of these vectors over the sequence indices 1…T.*

MLE state-estimation is achieved through the Viterbi algorithm (Viterbi). Viterbi exploits dynamic programming techniques to pick the state sequence with highest computed probability out of all possible state sequences in polynomial time. The **Appendix** contains a python implementation of Viterbi taken directly from Wikipedia and closely resembles the adapted implementation in FEL. [refer to 6.036 notes for more on viterbi]

2.3.2 Supervised model-estimation

Supervised Model-estimation reduces to computing statistics over training data (Sarkar 2013). Users provide training examples in the form (**x** = *raw text*, **y** = *extraction example*).

*(**x** = 15 Progenitorivox at $10.50 refilled for (james77@inbox.com, 8795552304) : 11/14/07,*
y *= {***date**: *11/14/07*, **email**: *james77@inbox.com*, **phone**: *8795552304*, **brand**: *Progenitorivox*, **packsize**: *15*, **price**: *10.50*}*)

Occurrence count probabilities are taken over training examples:
FEL requires only 1 training example to compute a model and typically less than 10 training examples are needed to extract common-case records. If training example requirement were not kept low, FEL would violate the Machine Driven, Human Guided KOP. Thus, FEL must cope with sparse training sets.

$$f(i, x, y) = \text{number of times } i \text{ is the initial state in } (x, y)$$
$$f(i, j, x, y) = \text{number of times } j \text{ follows i in } (x, y)$$
$$f(i, o, x, y) = \text{number of times } i \text{ is paired with observation } o \text{ in } (x, y)$$

$$\Pi_i = \frac{\sum_l f(i, x^{(l)}, y^{(l)})}{\sum_l \sum_k f(k, x^{(l)}, y^{(l)})}$$

$$A_{i,j} = \frac{\sum_l f(i, j, x^{(l)}, y^{(l)})}{\sum_l \sum_k f(i, k, x^{(l)}, y^{(l)})}$$

$$B_{i,j} = \frac{\sum_l f(i, o, x^{(l)}, y^{(l)})}{\sum_l \sum_{o' \in V} f(i, o', x^{(l)}, y^{(l)})}$$

Figure 18 : *Parameters of an HMM computed as normalized occurrence counts (Sarkar 2013).*

To overcome the unrepresentative limitations of a sparse data set, FEL's models undergo smoothing. Smoothing is covered in detail in **Section 3.5**.

2.3.3 Topology

The topology of an HMM is determined by its state set and its emission set.

To determine the number of states, FEL asks users for a list of field names for extraction. Each field name corresponds to a state in the underlying HMM. **Section 5.3.3** discusses Fast-State Discovery, an unsupervised state-learning approach.

In practice, picking a representative emission set can be accomplished by selecting a granular, fixed set of token types. Each token type is treated as a unique emission type. **Section 5.2.2** discusses Text Frequency - Inverse Document Frequency (TF-IDF) as a means to learn domain-specific token types from the text data.

2.4 Probabilistic flexibility

This section examines the flexibility of probabilistic modeling when encountering extra data or missing data.

Extra data is treated as noise that must be detected and subsequently ignored. **Section 2.4.1** explains FEL's approach for augmenting the underlying HMM to model "fields in a sea of noise".

FEL must operate on free-text records with missing information gracefully; FEL must not depend on the presence of fields and must indicate when a field was not found (via a **NULL** value). **Section 2.4.2** examines how FEL *smooths* its model to account for shortcuts in its language model for skipping over missing values.

FEL can be extended to handle typos and other date-entry errors gracefully via the Natural Language Processing (MLP) technique of *n-gram distances*, as covered in **Section 5.2.4**.

2.4.1 Extra data as noise

FEL's design is to treat free-text as *fields in a sea of noise*. All data that does not correspond to a field, instead corresponds to noise. For the HMM, FEL achieves this by augmenting the model with a dedicated *Noise state.*

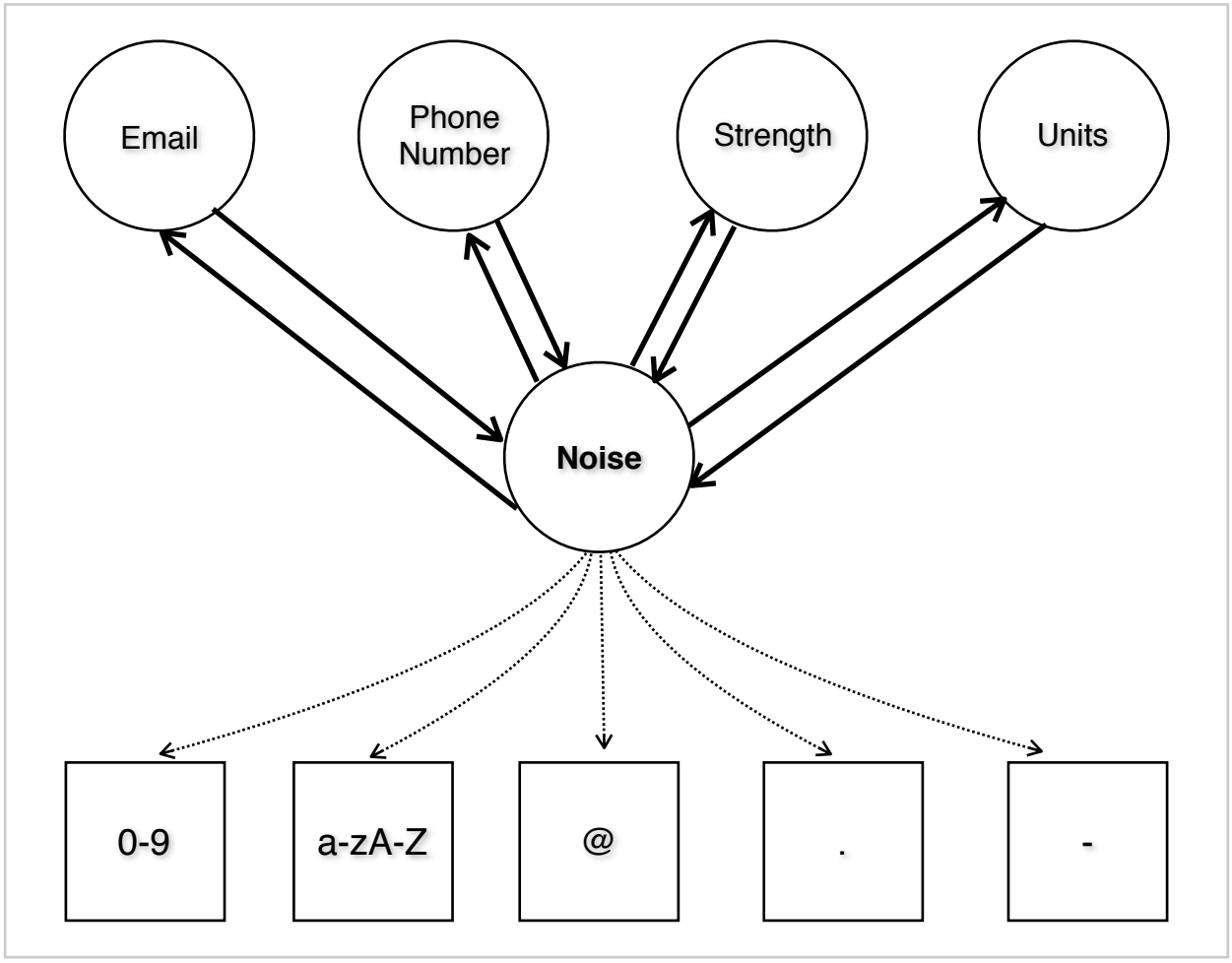


Figure 19 : *Noise modeled as an extra state in the HMM. A special property of the Noise state is that it should be able to emit any token type. Other transition and emission arrows/ probabilities omitted for brevity.*

Any pieces of free-text not included in the **y** label are assumed to be emissions from the noise state. Thus, parameters pertaining to the noise state do not receive any special treatment; they are set by the same learning algorithm. **Section 3.3** and **Section 3.4** discuss how to model noise more precisely in the hopes of increasing the fidelity of the model via *landmarks* and *links*.

2.4.2 Missing values as shortcut

Its possible that FEL will come across a free-text record whose field ordering does not correspond to a valid path through the state transitions in its HMM. In these cases, FEL performance would be poor in terms of quality.

The cause is that the supervised model-estimation algorithm only assigned non-zero probabilities to patterns that were seen in the sampled training set. The model needs a way to account for unseen patterns when they are inevitably encountered. To this end, FEL *smooths* its model.

$$\hat{\theta}_i = \frac{x_i + \alpha}{N + \alpha\delta}$$

Intuitively, additive smoothing allows FEL to control to what extent it views the learned patterns as hard rules vs. soft guidelines.

2.5 Advanced Content-Structure Models

Modeling fields as PMFs is sufficient when fields can be differentiated by the token types. In cases where token ordering plays a role in determining fields, the model requires another mechanism by which to capture differences in token ordering.

Fields are composed of smaller semantic blocks. Foe example, phone numbers are not just assortments of digits, periods, and dashes, but rather they have *country codes*, *area codes*, *extensions*, etc… These are terms humans commonly use to describe what a phone number *looks like*. To achieve this level of granularity within fields, FEL is designed to swap out PMF models for full-blown HMMs.

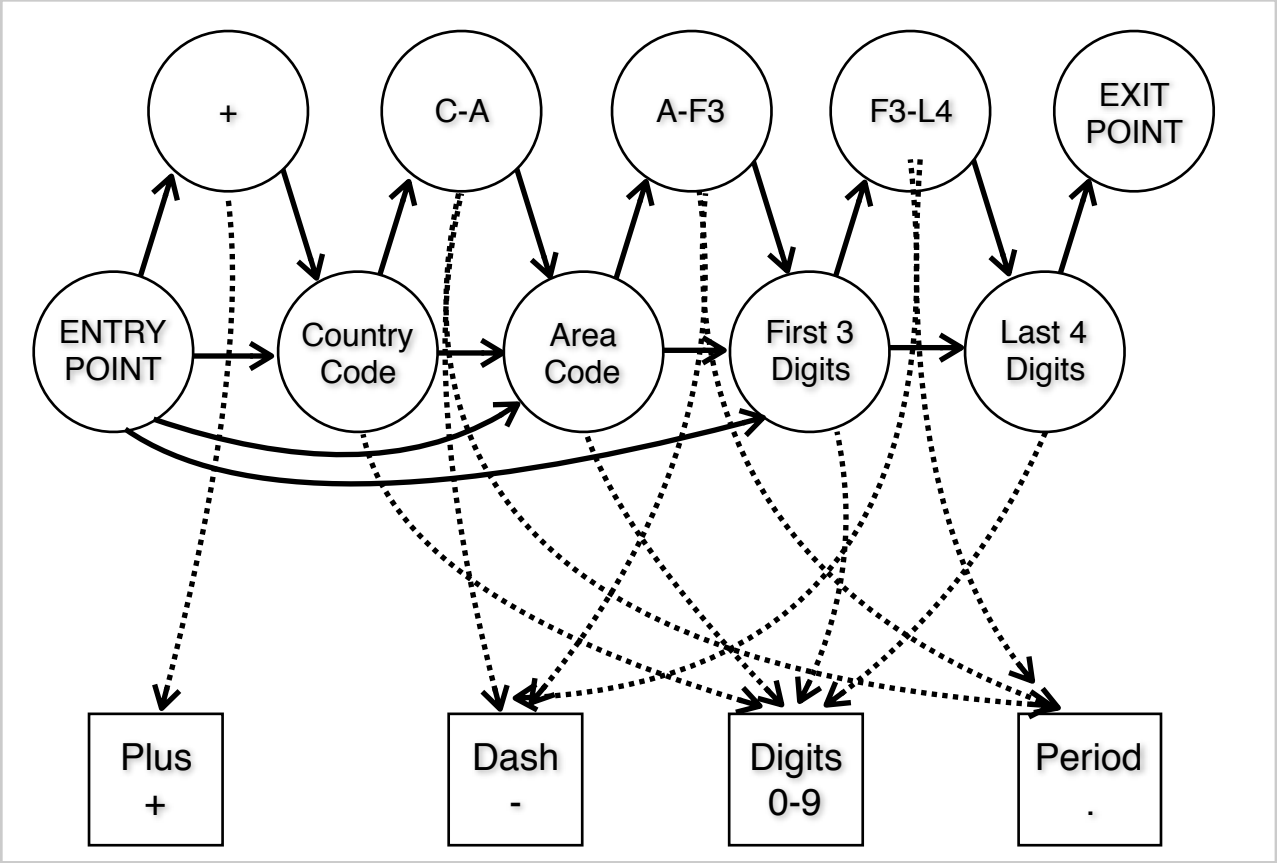


Figure 20 : *Phone number models as an HMM. Entry/Exit points added to chain field HMMs into a larger model. Intuitively, entry/exit points enforce that fields cannot start or finish in the middle, though this assumption can be softened via smoothing to cope with typos.*

By modeling fields with HMMs, the model can now distinguish between fields with ambiguous PMFs.

To combine our field models into 1 super-HMM, the model is augmented with entry and exit points for each field.

Entry and exit points capture the intuition that typically a field transition occur only once the previous field has ended.

One key advantage of modeling fields as fully qualified HMMs is that these sub-HMMs become Additionally, HMMs as field models means that these models can be reused across different datasets and projects. A team could build up a catalog of these pre-trained *reusable field detectors* to bootstrap new projects without needing to do any example labeling. **Section 5.3** discusses the challenges associated with the implementation of sub-HMMs as field models.

3. Implementation

FEL implements the model from **Section 2**, but it also includes a number of optimizations designed to make it work better in practice. In accordance to the Principled Learning KOP, I took special care to ensure that the modifications modeled new situations instead of hard-coding a fix.

The approach here is to implement the simplest version of FEL, find obstacles or bottlenecks in practice, modify FEL, and repeat.

Before any implementation iteration, the first step is to implement the tokenizer and the token types so that FEL works end-to-end.
In my subsequent experiments I identified 5 major changes to FEL: (1) extendible token types, (2) modeling *landmarks*, (3) modeling *links*, (4) smoothing, and (5) ambiguity detection. The rest of this section looks at each of these improvements in detail.

Pharmaceutical transactions
Pasceline D 16 vial(s) 22.5 mg/ml $168.83
Doloxan 24 pre-filled injection syringe - 5 mg/ml for $84.37
E-Z Doze It Sleeping Pills sold at $14.40 : 8 gel-soft pills 50 mg
Zydrate patch 1X500mg/CM^2 $8.97
11/17/2008 Medi-Gel 28 gel container(s) $21.55 (SALE 20% OFF) 40 MG/G
1 Jamitol/Ambrosia 40 mg strength cream pack at $11.15
$56.28 Hourai X 40mg (28 tablets)

Figure 21 : *Pharmaceutical free-text for transactions. Fields are: brand, pack size, form, strength, units, and price.*

Figure 21 shows the motivating example used throughout this section.

3.1 Bootstrapping Token Types

FEL employs a simple fixed set of token types, each corresponding to a regular expression (regex). Text is read by the tokenizer which finds the longest regex match among token types.

```
def tokenize(text, types):
    i = 0
    while i < len(text):
          longest_match = longest_match(text[i:], types)
          if longest_match:
                yield longest_match
                i += len(longest_match.string)
                continue

from collections import namedTuple
Token = namedTuple('Token', ['type', 'string'])

import re # regular expression library
def longest_match(text, types):
   longest_match = None
   for match in (re.match(t.regex, text) for t in types):
          if match:
                match = match.group()
                if longest_match is None \
                or len(match) > len(longest_match.string):
                      longest_match = Token(type=t.name,string=match)
```

Figure 22 : *FEL's tokenizer implemented in python.*

As recommended in the Distill paper[distill], I first tried using 3 simple token types: Numbers, Letters, and Symbols.

```
from collections import namedTuple
TokenType = namedTuple('TokenType', ['name', 'regex'])
TokenType(name='number', regex=r'\d+')
TokenType(name='letters', regex=r'[a-zA-Z]+')
TokenType(name='symbol', regex=r'''[-`=[\]\\;',./~!@#$%^&*()_+
{}|:"<>?\s]''') # TODO: extend for unicode support…
```

Figure 23 : *Numbers, Letters, and Symbols as token types based on regex patterns.*

With only 3 token types, FEL could differentiate trivial fields (eg. words and phone numbers), but quickly struggled with differentiating fields based on symbols.

Free-text: Doloxan 24 pre-filled injection syringe - 5 mg/ml for $84.37

Tokens: abc num abc sym abc abc abc sym num abc sym abc abc sym num sym num

Labeling: *brand:*Doloxan, *packsize:*24, *noise:*pre-filled injection, *form:*syringe, *noise:*-, *strength:*5, *units:*mg/ml, *noise:*for, *price:*$.84.37

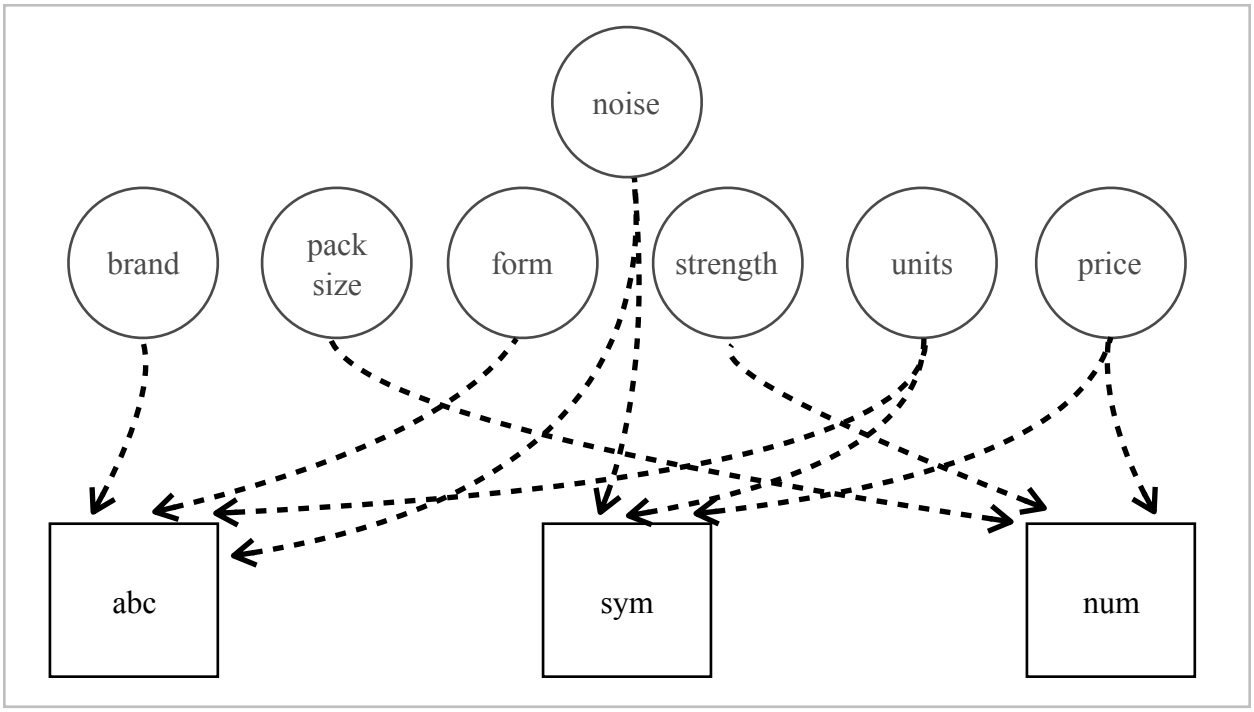


Figure 24 : *Pharmaceutical data tokenized into 3 tokens types: Numbers (num), Letters (abc), and Symbols (sym). At this point, there is ambiguity between noise and units as symbols are all lumped into one monolithic token type. Difference between pack-size and strength is also ambiguous. The ambiguity between brands and forms requires heavy-lifting via Natural Language Processing (NLP) and is covered in* ***Section 5.2****.*

3.2 Extendible Token Types

The token type set was to coarse-grained and did not capture the structural differences of fields. To address this problem, I made 2 key insights: (1) symbols have substantially different semantics and (2) the user may know some domain-specific terms that could be used as fine-grained token types.

This section discusses specific implementation changes based on these 2 insights. **Section 5.2** covers more improvements to FEL's tokenization.

3.2.1 Symbol Token Types

FEL's implementation does not include the advanced content structure models of **Section 2.5**, FEL cannot differentiate numerical data based on the token order, but can detect differences in token types. Therefore, I expanded FEL's symbol token type into more granular token types where each symbol is its own token type.

Free-text: Doloxan 24 pre-filled injection syringe - 5 mg/ml for $84.37

Tokens: abc num abc - abc abc abc - num abc / abc abc $ num . num

Labeling: *brand:*Doloxan, *packsize:*24, *noise:*pre-filled injection, *form:*syringe, *noise:*-, *strength:*5, *units:*mg/ml, *noise:*for, *price:*$.84.37

Figure 25 : *Pharmaceutical data tokenized with each symbol as its own token type. Units are now easily distinguished from noise via their PMFs.*

Preserving symbols through tokenization in this manner increased FEL's performance considerably to the point where tokenization was no longer the bottleneck.

3.2.2 User-Defined Token Types

While humans easily pick out domain-specific values (ie. special terminology), FEL's model could not yet detect them. **Section 5.2** discusses how FEL can be extended to *learn* domain-specific terms via the TF-IDF technique. For this section, FEL will be extended to allow users to specify domain-specific terms that they already know about.

Free-text: 1 Jamitol/Ambrosia 40 mg strength cream pack at $11.15

Tokens: num abc / abc num abc abc abc abc abc $ num . num

Labeling: *packsize:*1, *brand:*Jamitol/Ambrosia, *strength:40*, *units:*mg, *noise:*strength, *form:*cream, *noise:*pack at, *price:*$.84.37

Figure 26 : *The model from 3.2.1 would confuse the brand name for a unit. Humans could easily identify the unit as* **mg**.

Figure 26 shows an example where FEL confuses fields because it does not know about the special value **mg**. Users working with this dataset have background knowledge of units. To comply with the MDHG KOP, users should be able to communicate this background knowledge to FEL.

Free-text: 1 Jamitol/Ambrosia 40 mg strength cream pack at $11.15

Tokens: num abc / abc num mg abc abc abc abc $ num . num

Free-text: Doloxan 24 pre-filled injection syringe - 5 mg/ml for $84.37

Tokens: abc num abc - abc abc abc - num mg / ml abc $ num . num

Figure 27 : *With mg as a token type for exact matches of the string 'mg' and ml for 'ml', models can detect units more easily. Unit PMF might look like {mg: 0.62, ml: 0.21, /:17}.*

FEL is able to model user-defined token types to gain further insight into the patterns within the free-text data.

3.3 Modeling Landmarks

Consider the case where the user wishes to extract the strength amount, but not the unit. In this case, **unit** is not a field and thus the model does not contain a state whose PMF has a non-zero probability for the **mg** or **ml** tokens.

Free-text: 1 Jamitol/Ambrosia 40 mg strength cream pack at $11.15

Tokens: num abc / abc num mg abc abc abc abc $ num . num

Figure 28 : *The custom **mg** token is not part of any state model. As a result, FEL has no way to exploit custom **unit** tokens to detect **strength**.*

As **Figure 28** demonstrates, **mg** is not captured by any state and thus FEL cannot use it to distinguish between pack sizes and strengths (both have PMF where probability for **num** is 1). Even if the user does not care about part of the free-text for extraction output, it may still be useful for modeling the data they do care about. These states are call *landmarks* to distinguish them from the states corresponding to fields the user wants to extract.

There is a strong relationship between fields such as **strength** and **unit**; detecting these fields is easier when FEL can look for patterns involving both of them rather than each one separately. FEL's language model (ie. the HMMs state layer) can capture relationships between fields.

However, FEL's language model is limited to part-of-speech rules between **consecutive states**. In this example, the tokens for **strength** and **unit** appeared without any other tokens between them. **Section 3.4** introduces *links* as a way to bridge the gap between related states at a distance.

3.4 Modeling Links

Taking advantage of field locality like in the example given for **Section 3.3** only works if the two related states are consecutive.

owner: [r:Y, w:Y, x:Y], group: [r:Y, w:N, x:N], other: [r:N, w:N, x:N]

In this example, FEL needs to extract the execute permission (**x**) for the group users (**group**). There is noise between *group* and *x:N*. Since FEL uses a 2nd-order HMM, it can only relate consecutive states.

… *owner-landmark:* **owner** *noise:* **: [r:Y, w:Y,** *x-landmark:* **x:** *execute:* **Y**…
… *group-landmark:* **group** *noise:* **: [r:Y, w:N,** *x-landmark:* **x:** *execute:* **N**…
… *other-landmark:* **other** *noise:* **: [r:N, w:N,** *x-landmark:* **x:** *execute:* **N**…

Figure 29 : *State assignments Viterbi makes when using landmarks for* owner, group, other, *and* x. *FEL's language model cannot yet bridge the noise gap between the user landmarks and the execution landmark.*

As a result FEL cannot distinguish the (*x-landmark*, *execute)* pairs to determine which one corresponds to *group*. One solution is to use a higher order (nth-order) HMM, but we cannot tell ahead of time what order to use in the general case. Furthermore higher-order HMMs will need more data to accurately estimate their parameters and will be less performant in terms of computation time.

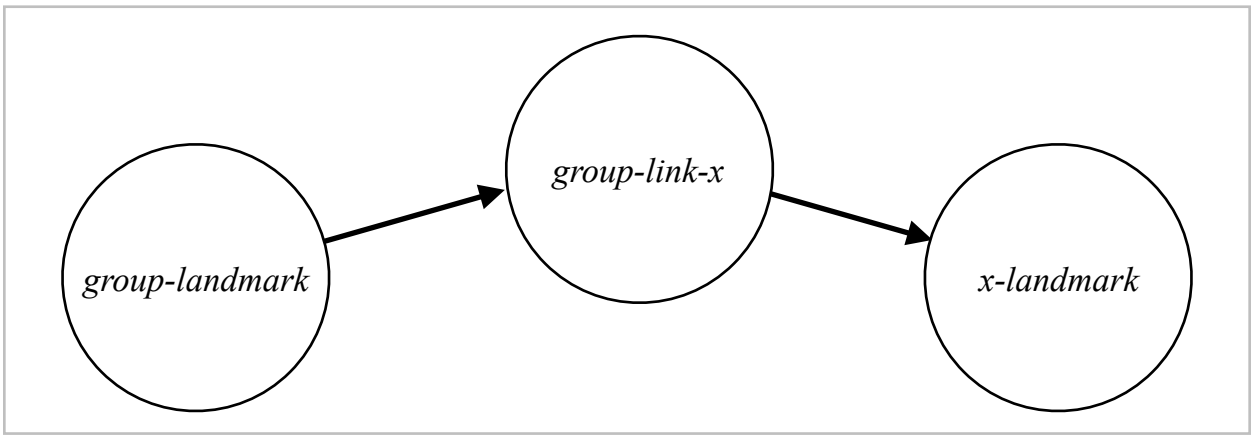
Instead, FEL introduces *links* as a way of modeling the gap between related fields:

links: (group-landmark → x-landmark)

This will result in:

… *group:* **group** *group-link-x:* **: [r:Y, w:N**, *x-landmark:* **x:** *execute:* **N** …

The link allows FEL to model a part-of-speech rule that bridges the gap between *group* and *x-landmark* via an additional state in the HMM which should be part of the path from *group* to *x-landmark*.



3.5 Smoothing

FEL represents a probabilistic model of the data seen in training examples. However, any patterns not exposed during training are assigned a probability of 0. Intuitively, FEL views training examples as an authoritative set of patterns which leads FEL to feel more like a set of rules than a probabilistic model. To model non-zero probabilities for unseen events, FEL employs *additive smoothing*.

```
1 import numpy as np
2 def normalize(M, smoothing=0, axis=0):
3   '''Additive smoothing
4   M -- the numpy array/matrix to be normalized
5   smoothing -- additive smoothing paramater α
6   axis -- specifices the axis along which to normalize
7   '''
8   def smooth(v):
9     return (v + smoothing) / (sum(v) + len(v) * smoothing)
10  return np.apply_along_axis(smooth, axis, M)
```

FEL applies smoothing to its 3 matrices A, B, and π. As such, FEL considers transitions between any two states, emission of any token from any state, and starting states for a given sequence with low but non-zero probabilities. As a result, FEL gracefully handles new language patterns (eg. unseen field orderings) and new content-structure patterns (eg. unseen forms for a given field or typos).

FEL's implementation broadly applies the same smoothing parameter α to the 3 matrices. **Section 5.3.4** covers how FEL could be extended to apply targeted smoothing so that different transitions, emissions, etc… are each smoothed differently.

3.6 Ambiguity Detection

To detect fields, FEL must be able to distinguish them from one another. Fields have 2 key differentiators: (1) transitions for that field's state (2) emission produce by that state. FEL uses the L1 norm as a similarity metric along with a threshold for that similarity metric to detect ambiguities.

3.6.1 Transition ambiguities

States have both incoming and outgoing transition probabilities. Either of these can differentiate the state.

```
1 def transition_score(self, i, j):
2    # outgoing transitions for i
3    trans_i_k = self.hmm.trans[i,:]
4    # incoming transitions for i
5    trans_k_i = self.hmm.trans[:,i]
6
7    # outgoing transitions for j
8    trans_j_k = self.hmm.trans[j,:]
9    # incoming transitions for j
10   trans_k_j = self.hmm.trans[:,j]
11
12   score = min(
13        sum(abs(trans_i_k - trans_j_k)),
14        sum(abs(trans_k_i - trans_k_j))
15   )
16   return score
```

Figure 30 : *FEL calculates transition scores by using the L1 norm. Note that FEL takes the minimum of the L1 norms calculated for incoming and outgoing transition probabilities as either one can differentiate the state.*

FEL compares the scores to a hard-coded threshold. If the score is under the threshold, the L1 norm indicates ambiguity due to transition similarity. Intuitively, this determines if the language model is able to differentiate states.

3.6.2 Emission ambiguities

Emission ambiguities are analogous to transition probabilities. Again, FEL uses the L1 norm and compares scores to a threshold to determine if there is ambiguity between states.

```
1. def emission_score(self, i, j):
2.   return sum(abs(self.hmm.emit[i,:] - self.hmm.emit[j, :]))
```

Intuitively, this determines if the content-structure can differentiate states.

3.6.3 Reporting ambiguities

If a pair of fields cannot be differentiated by either the transition scores or the emission scores, it is marked as an ambiguity. FEL can then warn the user of this ambiguity and request further guidance in order to resolve the ambiguity.

4 TXTract Application

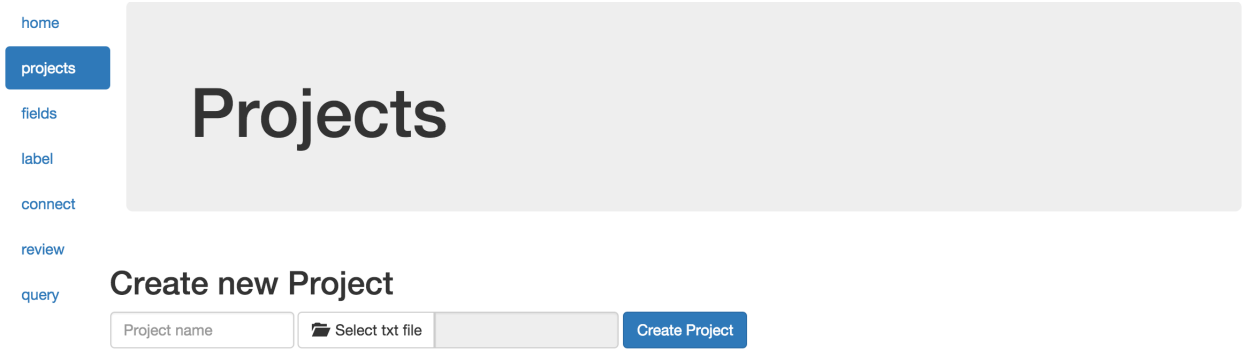
In this section, I present TXTract, a light-weight web application built on top of FEL. TXTract is a minimal UI that adds persistence through a database, enabling users to save their work between sessions and to seamlessly switch between multiple on-going projects.

4.1 Workflow

TXTract imposes a workflow for users to follow and to provide human guidance to FEL according to the Machine Drive, Human Guided KOP. The workflow consists of 4 major steps: (1) create a project, (2) specify field names, (3) label examples, and (4) extract. Along the way TXTract will make use of FEL's ambiguity detection to prompt the user for more feedback.

The rest of this section aims to provide more detail on the 4 steps of the workflow.

4.1.1 Create a project



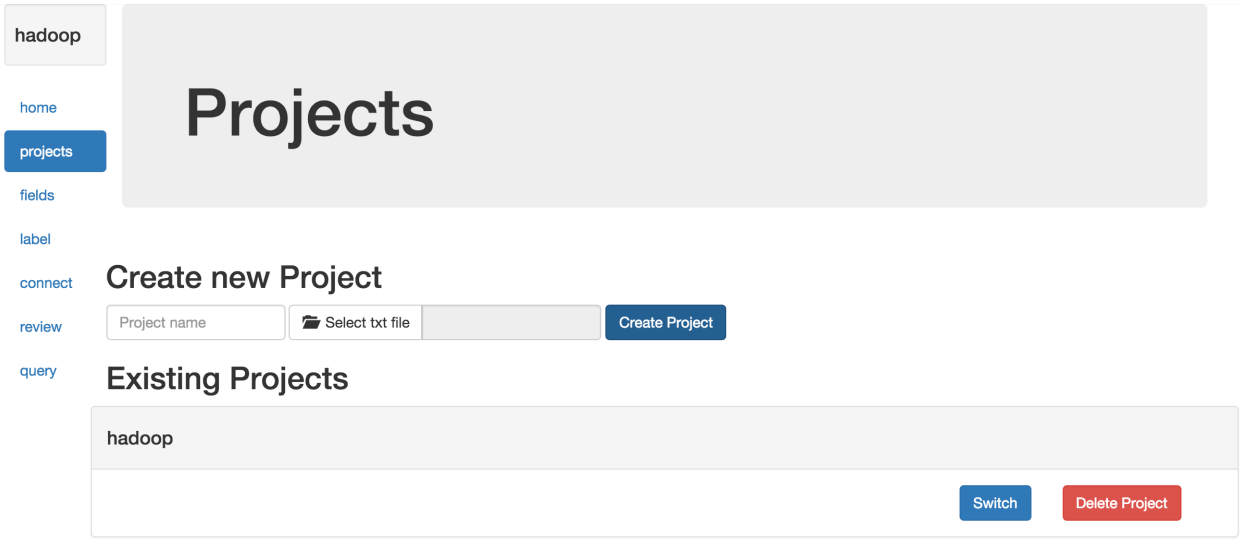
Users begin by creating a project. To create a project, users must specify a project name and must register the text file to undergo the extraction process.

When the project is created the system will create a project directory to house all the data for the new project. The project directory will contain the project's metadata file, as well as the data for the underlying database.

New projects may be created at any time. The system only requires that projects names be unique.

4.1.2 Switching and Deletion

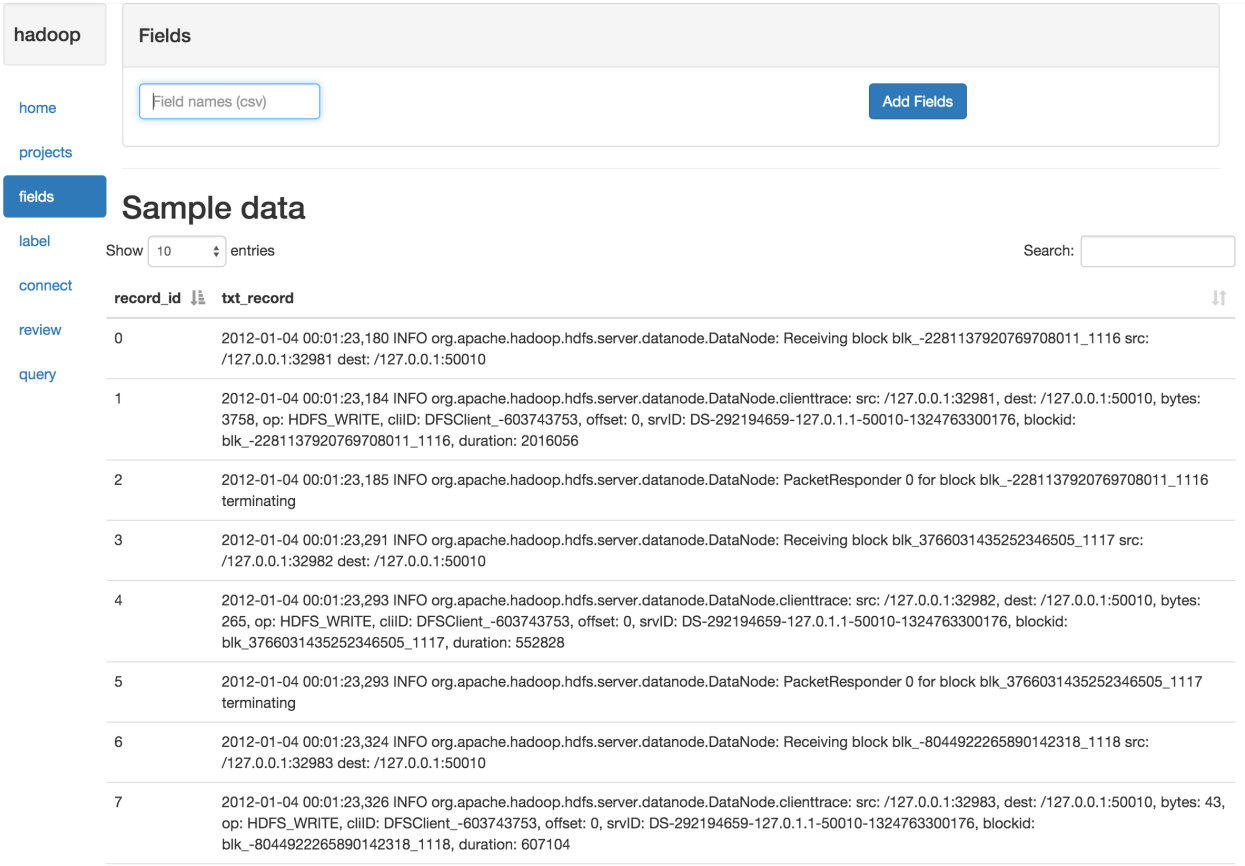
Once a project has been created, users may switch to work on that project by selecting the **Switch** option for that project in the Projects page. The current project can be determined by looking at the label on the top left of any page.



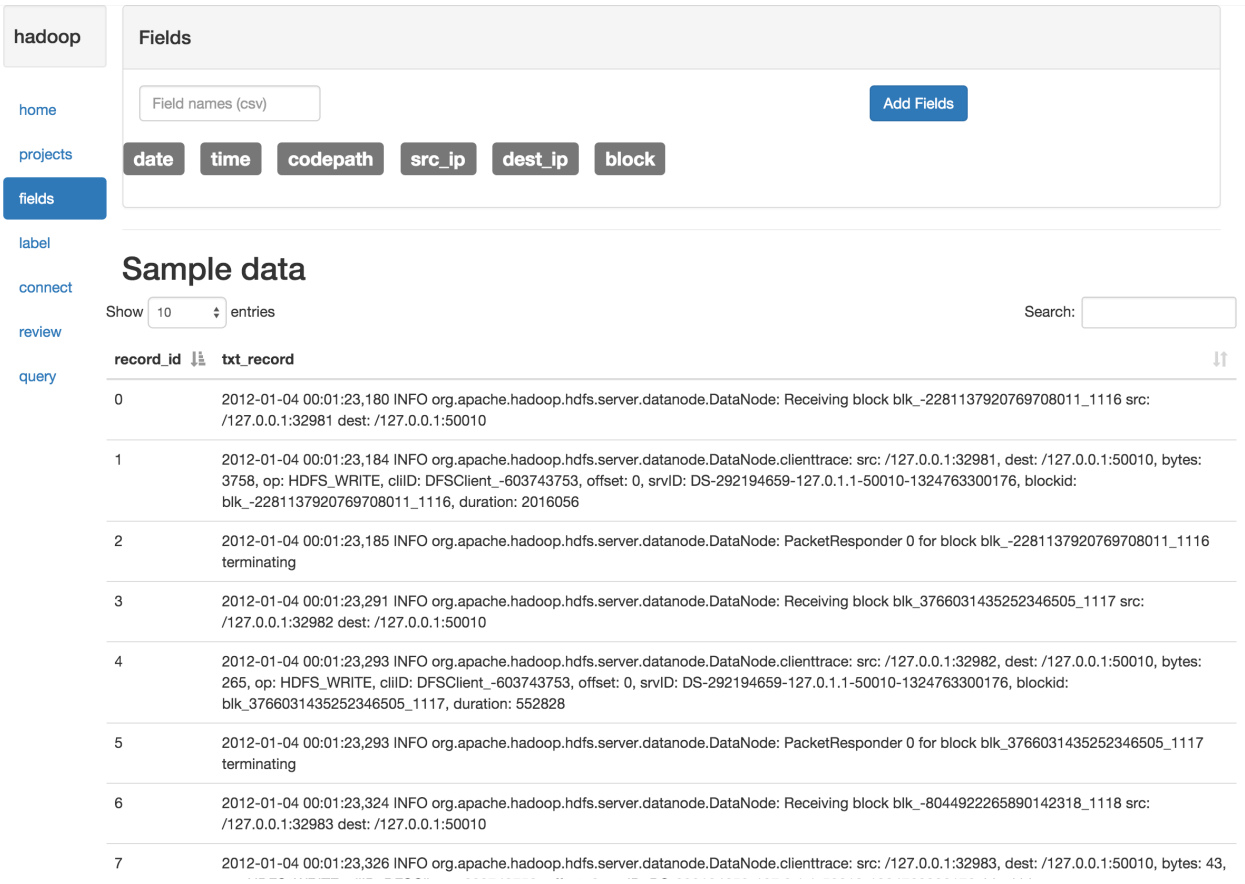
The data in all the pages will correspond to the currently selected project.

At any point in the process, users may delete existing projects by navigating to the Projects page and selecting **Delete Project**. This removes the project's directory and all of its content permanently.

4.1.3 Choose field names

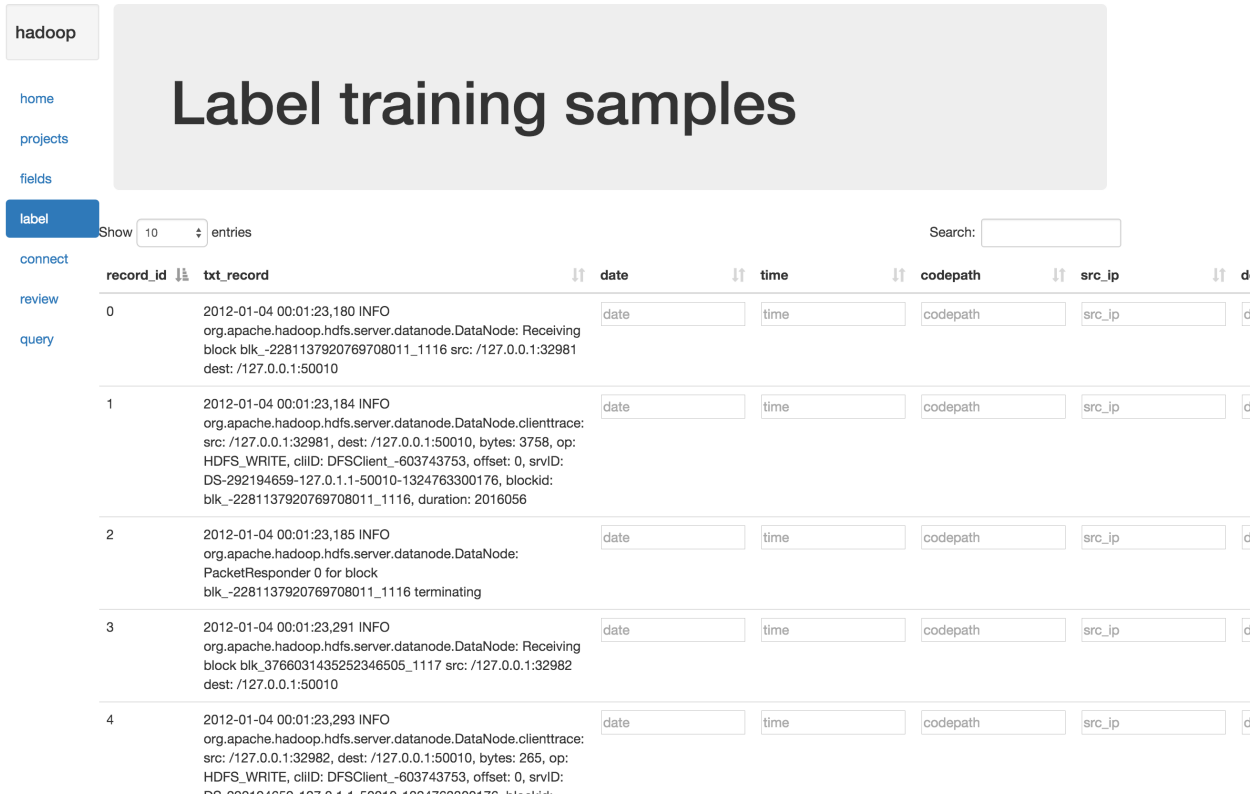


Next, users specify the names of the fields they wish to extract. The names can be arbitrary; the system does not use the names in any of the Machine Learning algorithms. They merely provide a convenient way to refer to the fields. For example, specifying **phone** as a field name will not cause the system to look for numerical entities automatically; Relating the field name to the type of content for that field is done in the labeling step.



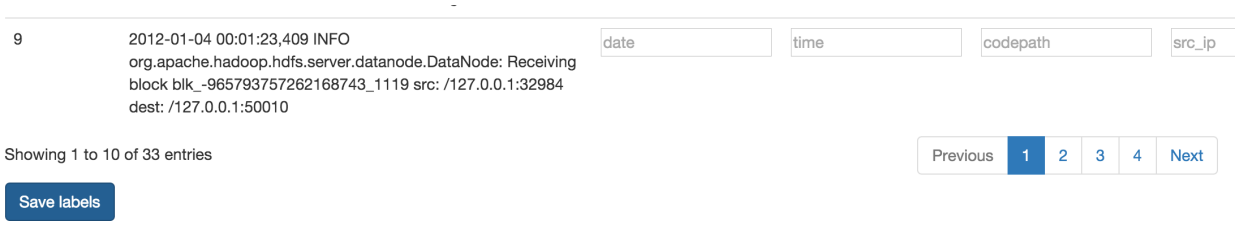
Fields specified here will be used in conjunction with any landmarks and links created in the *Connect* page to determine the number of states for the underlying Hidden Markov Model.

4.1.4 Label training examples



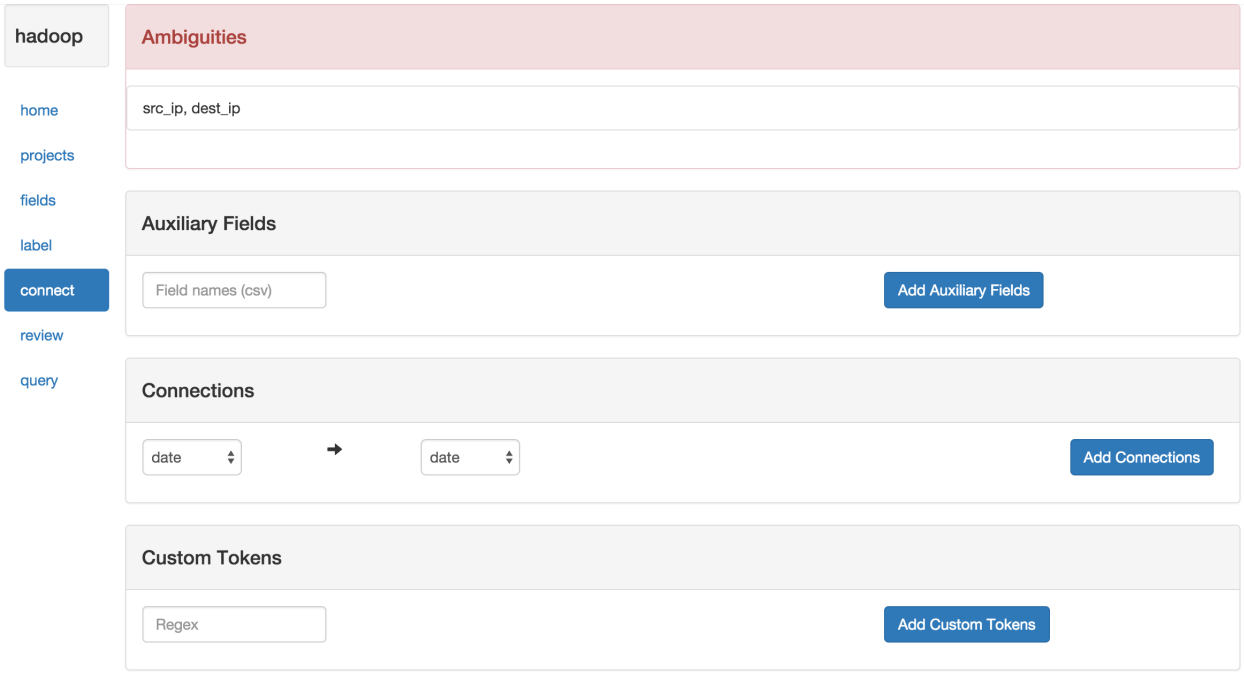
Users identify the substrings that correspond to the fields specified in the previous step. These will be tokenized and fed to FEL's supervised machine learning algorithm for estimating parameters of the Hidden Markov Model.

txt_record	date	time	codepath	src_ip	dest_ip	block
2012-01-04 00:01:23,180 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-2281137920769708011_1116 src: /127.0.0.1:32981 dest: /127.0.0.1:50010	2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32981	127.0.0.1:50010	2281137920769708011
2012-01-04 00:01:23,184 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32981, dest: /127.0.0.1:50010, bytes: 3758, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_-2281137920769708011_1116, duration: 2016056	2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32981	127.0.0.1:50010	2281137920769708011
2012-01-04 00:01:23,185 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_-2281137920769708011_1116 terminating	2012-01-04	00:01:23	org.apache.hadoop.hdf	src_ip	dest_ip	2281137920769708011
2012-01-04 00:01:23,291 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_3766031435252346505_1117 src: /127.0.0.1:32982 dest: /127.0.0.1:50010	2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32982	127.0.0.1:50010	31435252346505_1117



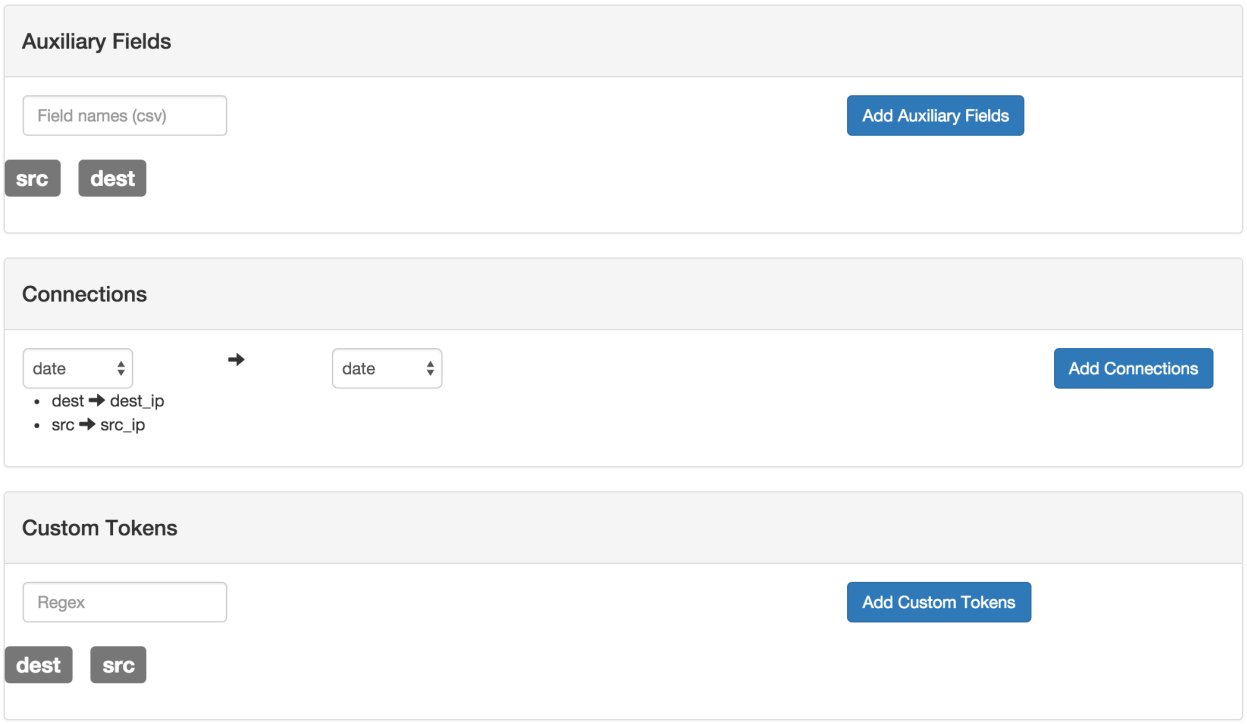
4.1.5 Resolve model ambiguities

TXTract uses FEL's ambiguity detection functionality to prompt users of any ambiguities so far.



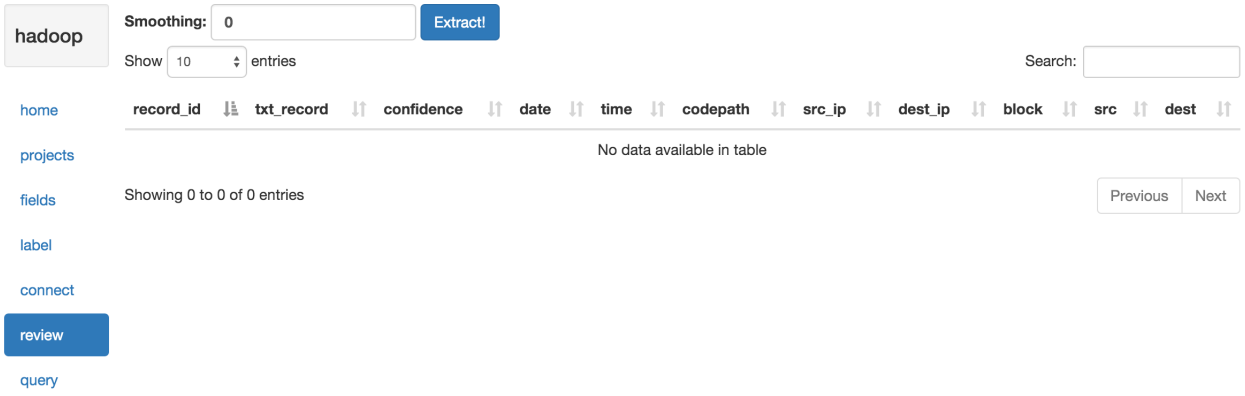
Fields can be disambiguated in 3 ways:

1. Modeling landmarks fields.
2. Providing links between fields to model the gaps between fields more precisely
3. Improving tokenization (in this case by allowing for the user to specify custom token tokens)



date	time	codepath	src_ip	dest_ip	block	src	dest
2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32981	127.0.0.1:50010	2281137920769708011	src	dest
2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32981	127.0.0.1:50010	2281137920769708011	src	dest
2012-01-04	00:01:23	org.apache.hadoop.hdf	src_ip	dest_ip	2281137920769708011	src	dest
2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32982	127.0.0.1:50010	3766031435252346505	src	dest

4.1.6 Extract results / review



Smoothing is applied to the model to coerce any deterministic parts of the model to their probabilistic counterparts. FEL then infers the best extraction for a given piece of text and TXTract outputs the results in a table.

txt_record	confidence	date	time	codepath	src_ip	dest_ip	bl
2012-01-04 00:01:23,180 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-2281137920769708011_1116 src: /127.0.0.1:32981 dest: /127.0.0.1:50010	3.73980316945e-50	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	[u'127.0.0.1:32981']	[u'127.0.0.1:50010']	[u'
2012-01-04 00:01:23,184 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32981, dest: /127.0.0.1:50010, bytes: 3758, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_-2281137920769708011_1116, duration: 2016056	1.13664076067e-105	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace']	[u'127.0.0.1:32981']	[u'127.0.0.1:50010']	[u'
2012-01-04 00:01:23,185 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_-2281137920769708011_1116 terminating	1.84902779147e-35	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	None	None	[u'
2012-01-04 00:01:23,291 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_3766031435252346505_1117 src: /127.0.0.1:32982 dest: /127.0.0.1:50010	4.24080433853e-48	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	[u'127.0.0.1:32982']	[u'127.0.0.1:50010']	[u'
2012-01-04 00:01:23,293 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32982, dest: /127.0.0.1:50010, bytes: 265, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_3766031435252346505_1117, duration: 552828	1.28891036528e-103	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace']	[u'127.0.0.1:32982']	[u'127.0.0.1:50010']	[u'
2012-01-04 00:01:23,293 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_3766031435252346505_1117 terminating	2.09673202701e-33	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	None	None	[u'
2012-01-04 00:01:23,324 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving	3.73980316945e-50	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	[u'127.0.0.1:32983']	[u'127.0.0.1:50010']	[u'

Description of the columns in the output ordered the same way as the screenshots:

1. **record_id**: an id assigned by the model to keep track of this piece of free-text
2. **txt_record**: The original, unprocessed text
3. **confidence**: unnormalized confidence score. **Section 5.4** discusses how these may be normalized to more closely resemble confidences rather than raw probabilities.

4.1.7 Smooth

Without smoothing, TXTract may output non-sensical results. This behavior is best described as FEL **forcing** the data to match a pattern learned from training.

2012-01-04 00:01:24,862 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling block blk_-2289356614634084790_1114 file /hadoop/dfs/data/current/blk_-2289356614634084790 for deletion	0.0	None	None	None	None	[u'2012-01-04 00:01:24, org.apache.hadoop.hdfs Scheduling block blk_-2 /hadoop/dfs/data/curren for deletion\n']
2012-01-04 00:01:24,862 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling block blk_6186891375771801408_1115 file /hadoop/dfs/data/current/blk_6186891375771801408 for deletion	0.0	None	None	None	None	[u'2012-01-04 00:01:24, org.apache.hadoop.hdfs Scheduling block blk_61 /hadoop/dfs/data/curren deletion\n']
2012-01-04 00:01:24,863 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Deleted block blk_-2289356614634084790_1114 at file /hadoop/dfs/data/current/blk_-2289356614634084790	0.0	None	None	None	None	[u'2012-01-04 00:01:24, org.apache.hadoop.hdfs Deleted block blk_-2289 /hadoop/dfs/data/curren
2012-01-04 00:01:24,863 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Deleted block blk_6186891375771801408_1115 at file /hadoop/dfs/data/current/blk_6186891375771801408	0.0	None	None	None	None	[u'2012-01-04 00:01:24, org.apache.hadoop.hdfs Deleted block blk_6186E /hadoop/dfs/data/curren

To overcome these hurdles, users can specify the smoothing parameter at the top of the page

2012-01-04 00:01:24,862 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling block blk_-2289356614634084790_1114 file /hadoop/dfs/data/current/blk_-2289356614634084790 for deletion	2.43909687879e-72	[u'2012-01-04']	[u'00:01:24']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode', u'/hadoop/dfs/data/current/blk']	None	None
2012-01-04 00:01:24,862 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling block blk_6186891375771801408_1115 file /hadoop/dfs/data/current/blk_6186891375771801408 for deletion	5.4099535517e-69	[u'2012-01-04']	[u'00:01:24']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode', u'/hadoop/dfs/data/current/blk']	None	None
2012-01-04 00:01:24,863 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Deleted block blk_-2289356614634084790_1114 at file /hadoop/dfs/data/current/blk_-2289356614634084790	4.60989414211e-71	[u'2012-01-04']	[u'00:01:24']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode', u'/hadoop/dfs/data/current/blk']	None	None
2012-01-04 00:01:24,863 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Deleted block blk_6186891375771801408_1115 at file /hadoop/dfs/data/current/blk_6186891375771801408	1.02248145221e-67	[u'2012-01-04']	[u'00:01:24']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode', u'/hadoop/dfs/data/current/blk']	None	None

5. Future Work

5.1 Record slicing

Record slicing refers to splitting a large piece of free-text into smaller pieces corresponding to records. In practice, free-text is often already sliced into records so FEL does not attempt to solve the corner cases where this is not the case.

Database tables with free-text columns are one example of real data that is already split. Even if the data is not pre-split, many datasets use record separators like newlines or some specific marker like "======" to denote record boundaries.

For the general case, FEL could be extended to model a **RECORD BOUNDARY** state in its HMM. Users then mark examples of **RECORD BOUNDARY** along with the other fields during training and FEL will learn to identify record boundaries.

5.1.1 Record types

One dataset may contain different record types. Eg. a log file may contain transactional information as well as stack traces. FEL could be modified to detect different record types as different paths through its HMM states and fan out the extractions to multiple output tables.

5.2 Tokenization

FEL's tokenization currently depends on a fixed set of token types chosen by experimenting with what works in practice. To fulfill the Principled Learning KOP, this section gives approaches for dynamic tokenization based on the patterns in the data as well as user feedback.

5.2.1 Near-miss token hierarchy

Instead of accepting the token type set currently in FEL, users could provide feedback to converge on a token type set on a per-project basis. Specifically, FEL could adopt a Near-miss hierarchy (Winston 1970) that started with coarse-grained token types. Users then specify important characters in the free-text data. FEL takes this feedback and splits a token type into more granular token types accordingly.

For example, consider **Symbols** as a coarse-grained token type. If users specify that periods and forward-slashes are important characters, FEL could split **Symbols** into **Punctuation** and **Math Operations**.

5.2.2 TF-IDF

In accordance with the Principled Learning KOP, FEL needs a way to *learn* what set of token types to use. Natural Language Processing (NLP) offers one possible solution via Text-Frequency Inverse-Document-Frequency (TF-IDF). Intuitively, TF-IDF enables FEL to measure the importance of pieces of text within the dataset.

Chunks of text considered important could then be used as the basis for token types. For example, if the word **from** appears in nearly every piece of free-text it can be modeled with an exact-match token on the word **from**. However, if a value like **Tylenol** appears in 8% of the data, it can be considered an important value for a field like **brand**, but not a landmark itself.

5.2.3 Incorporating Preprogrammed Language Models

FEL's current implementation does not make use of any preprogrammed language models. However, existing language models may provide good features from which to create token types. For example, an English language model can help FEL identify English words and mark non-

English words like **Tylenol** as likely proper nouns and therefore as specific values for proper noun fields like **brand**.

5.2.4 Typo n-grams

FEL implements smoothing to coerce its model into a probabilistic model capable of detecting fields in the presence of noise and errors. Smoothing can enable FEL to detect typos in some cases. However, Natural Language Processing (NLP) also offers n-grams as a potential solution to this problem. FEL's typo resolution capabilities would be augmented by acting on computed n-gram similarity scores.

5.3 Implementing field detectors as HMMs

FEL models fields with PMF models that do not capture token ordering. As discussed in **Section 2.5**, FEL could be extended to support full HMMs as field detectors. Combining these HMMs is simply done by adding transitions from field exit points to field entry points.

5.3.1 Entry/Exit points

Conceptually, entry and exit points are straightforward as described above. However, implementing entry and exit points requires rethinking FEL's internals as currently the code depends on the assumption that each state emits some token. Unfortunately, the quick-fix of an empty string token for entry and exit points will not work without further modifications; an empty string token could match any point in the free-text, not just field boundaries.

5.3.2 Reusability

As an added benefit, HMM field detectors should be accurate enough to warrant reuse across projects. To implement this reusability, FEL needs to add a notion of persistence for models.

Furthermore, an HMM field detector is coupled with a specific set of token types. Bringing in multiple HMM field detectors necessitates a method for resolving conflicts between sets of token types.

5.3.3 FSD

Modeling fields as HMM means that each field needs its own topology. Up to this point, determine topology is FEL has been done by asking the user for field names. Analogously, FEL could ask the user for the states in a field HMM, but this does not bode well; for complex data set there may be upwards of 100 fields. Additionally, users need to describe the structure of a field rather than simply provide examples of the field.

To alleviate these burdens, FEL could be extended to take advantage of Fast State Discovery (FSD). Fast State Discovery is an unsupervised algorithm for learning the number of states in an HMM given the observable emission set (Siddiqi et al. 2007).

5.3.4 Targeted smoothing

Smoothing is applied globally in FEL; the same smoothing parameter α is used to smooth every row of the 3 HMM matrices.

To extend FEL, smoothing should be more targeted. For example, the **Noise** state should be smoothed heavily so that it can accurately model any data. State models made with only a few data points should be aggressively smoothed as well, whereas state models derived from large amounts of data only need small amounts of smoothing. Intuitively, smoothing is used to overcome sparse data so FEL should apply smoothing accordingly.

5.4 Confidence

FEL reports raw state sequence probabilities as confidences. After smoothing, all probabilities will be less than 1 which ensures that the probability decreases as the sequence is made longer. In other words, the probabilities are not comparable without normalizing by length. Thus, to make these probabilities more like confidence scores, FEL must normalize them.

Additionally, FEL has no mechanism for reporting the confidence of each field but only for the entire sequence. To extend FEL, FEL could find the probability of the same state sequence with the field replaced with noise or with other fields. These replacement probabilities could then be combined to produce a confidence on a per-field basis.

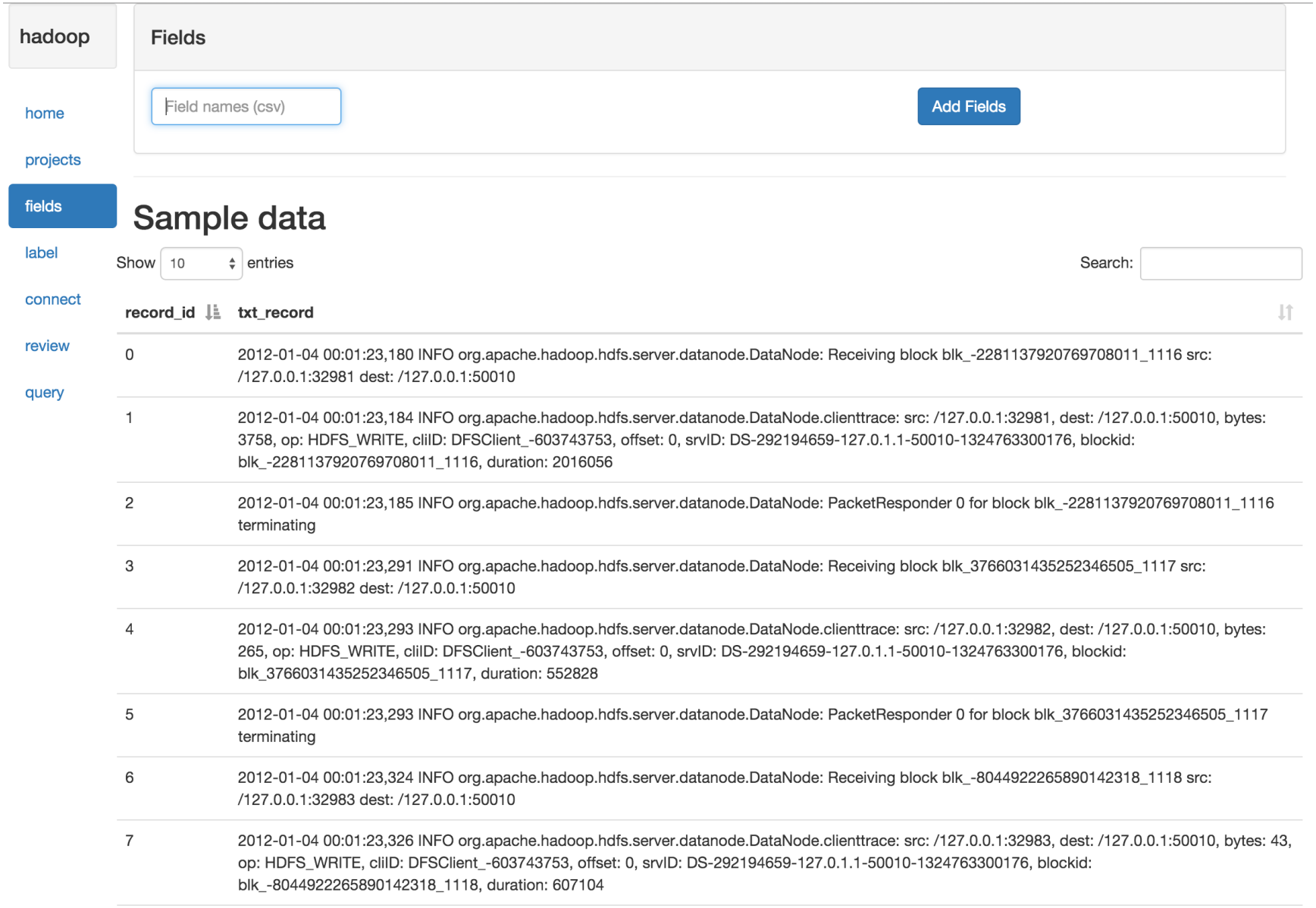
Finally, FEL could be extended to **generate** field from its model to expose any inconsistencies to the user. For example, a PMF model may be able to detect phone numbers accurately, but without a notion of token ordering the PMF model may generate data like ".3.6.+211-489". Generating data is a stricter test of semantic understanding than detection.

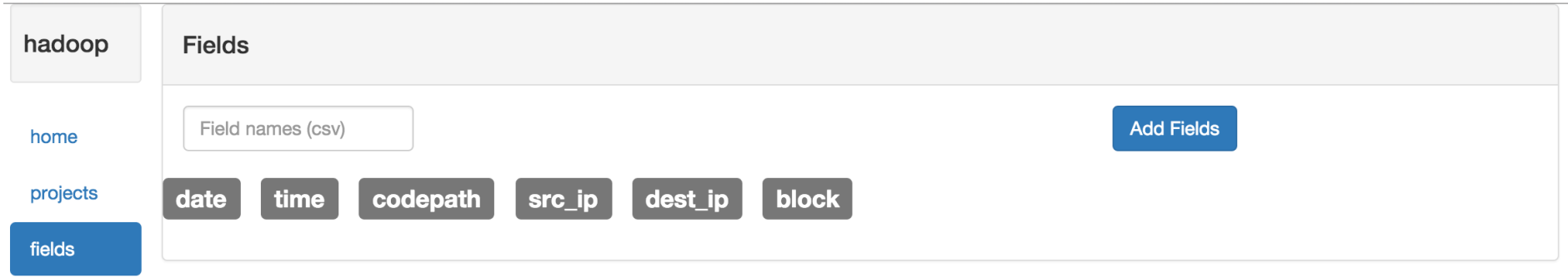
6. Appendix

```
def viterbi(obs, states, start_p, trans_p, emit_p):

    # Run Viterbi for t > 0
    for t in range(1, len(obs)):
        V.append({})
        newpath = {}

        for y in states:
            (prob, state) = max(
                (V[t-1][y0] * trans_p[y0][y] * emit_p[y][obs[t]], y0)
                for y0 in states
            )
            V[t][y] = prob
            newpath[y] = path[state] + [y]

        # Don't need to remember the old paths
        path = newpath

    # Return the most likely sequence over the given time frame
    n = len(obs) - 1
    (prob, state) = max((V[n][y], y) for y in states)
    return (prob, path[state])
```

```
def tokenize(self, text, stop=True):
    i = 0
    while i < len(text):
      for types in [self.user_defined_types, self.builtin_types]:
            longest_match = longest_match(text[i:], types)
            if longest_match:
                  yield longest_match
                  i += len(longest_match.string)
                  continue
      yield Token.unknown()
        i += 1

    if stop:
      yield Token.stop()

import re # regular expression library
def longest_match(text, types):
      longest_match = None
      for match in (re.match(t.regex, text) for t in types):
            if match:
                  match = match.group()
                  if longest_match is None \
                  or len(match) > len(longest_match.string):
                        longest_match = Token(string=match,type=t.name)
```

```
import numpy as np
def normalize(M, smoothing=0, axis=0):
  '''Additive smoothing
  M -- the numpy array/matrix to be normalized
  smoothing -- additive smoothing paramater α
  axis -- specifices the axis along which to normalize
  '''
  def smooth(v):
    return (v + smoothing) / (sum(v) + len(v) * smoothing)
  return np.apply_along_axis(smooth, axis, M)
```


hadoop
home
projects
fields
label
connect
review
query
Fields
Field names (csv)
Add Fields
Sample data
Show 10 entries
Search:
record_id
txt_record
0
2012-01-04 00:01:23,180 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-2281137920769708011_1116 src: /127.0.0.1:32981 dest: /127.0.0.1:50010
1
2012-01-04 00:01:23,184 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32981, dest: /127.0.0.1:50010, bytes: 3758, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_-2281137920769708011_1116, duration: 2016056
2
2012-01-04 00:01:23,185 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_-2281137920769708011_1116 terminating
3
2012-01-04 00:01:23,291 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_3766031435252346505_1117 src: /127.0.0.1:32982 dest: /127.0.0.1:50010
4
2012-01-04 00:01:23,293 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32982, dest: /127.0.0.1:50010, bytes: 265, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_3766031435252346505_1117, duration: 552828
5
2012-01-04 00:01:23,293 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_3766031435252346505_1117 terminating
6
2012-01-04 00:01:23,324 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-8044922265890142318_1118 src: /127.0.0.1:32983 dest: /127.0.0.1:50010
7
2012-01-04 00:01:23,326 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32983, dest: /127.0.0.1:50010, bytes: 43, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_-8044922265890142318_1118, duration: 607104



Sample data

Show 10 entries

Search:

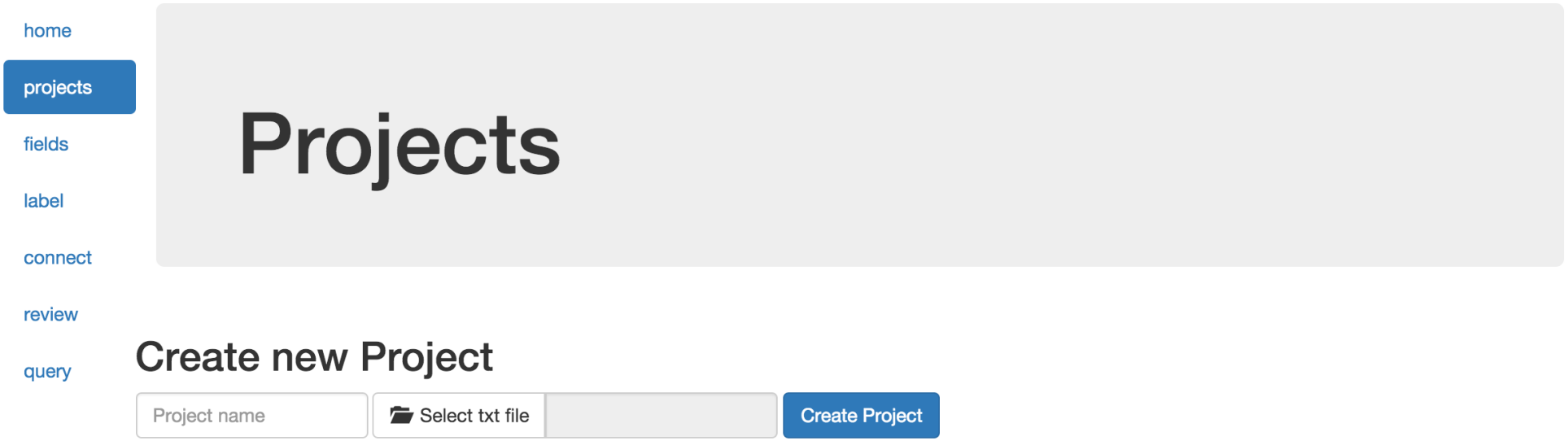
record_id	txt_record
0	2012-01-04 00:01:23,180 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-2281137920769708011_1116 src: /127.0.0.1:32981 dest: /127.0.0.1:50010
1	2012-01-04 00:01:23,184 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32981, dest: /127.0.0.1:50010, bytes: 3758, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_-2281137920769708011_1116, duration: 2016056
2	2012-01-04 00:01:23,185 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_-2281137920769708011_1116 terminating
3	2012-01-04 00:01:23,291 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_3766031435252346505_1117 src: /127.0.0.1:32982 dest: /127.0.0.1:50010
4	2012-01-04 00:01:23,293 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32982, dest: /127.0.0.1:50010, bytes: 265, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_3766031435252346505_1117, duration: 552828
5	2012-01-04 00:01:23,293 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_3766031435252346505_1117 terminating
6	2012-01-04 00:01:23,324 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-8044922265890142318_1118 src: /127.0.0.1:32983 dest: /127.0.0.1:50010
7	2012-01-04 00:01:23,326 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32983, dest: /127.0.0.1:50010, bytes: 43,

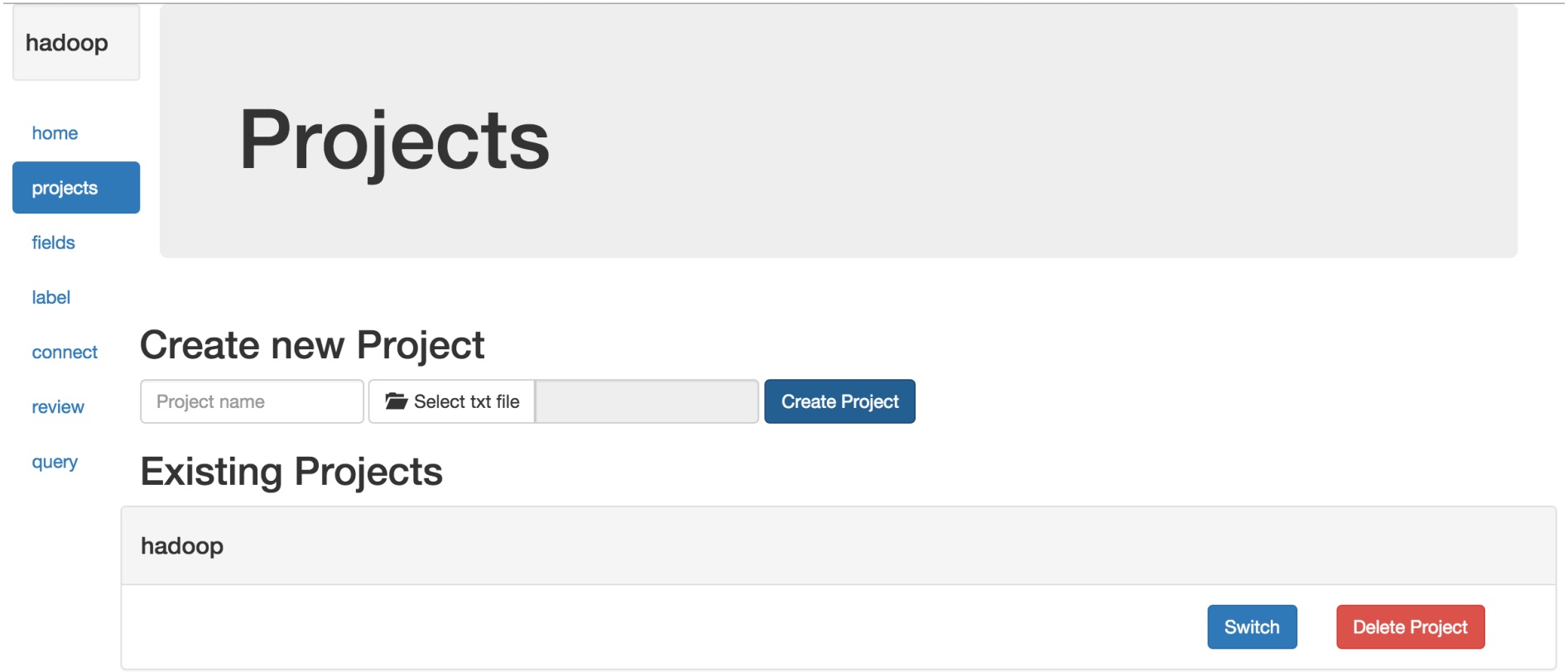
Pharmaceutical transactions
Pasceline D 16 vial(s) 22.5 mg/ml $168.83
Doloxan 24 pre-filled injection syringe - 5 mg/ml for $84.37
E-Z Doze It Sleeping Pills sold at $14.40 : 8 gel-soft pills 50 mg
Zydrate patch 1X500mg/CM^2 $8.97
11/17/2008 Medi-Gel 28 gel container(s) $21.55 (SALE 20% OFF) 40 MG/G
1 Jamitol/Ambrosia 40 mg strength cream pack at $11.15
$56.28 Hourai X 40mg (28 tablets)

brand	packsize	form	strength	units	price
Pasceline D	16	vial	22.5	mg/ml	$168.83
Doloxan	24	syringe	5	mg/ml	$84.37
E-Z Doze It Sleeping Pills	8	pills	50	mg	$14.40
Zydrate	1	patch	500	mg/CM^2	$8.97
Medi-Gel	28	gel	15	MG/G	$56.28
Jamitol Ambrosia	1	cream	1	G	$11.15
Hourai X	50	tablets	40	mg	$21.55

Prescription Pick-Up
On 10/3/2009 benbitd@mit.edu (+1-470-555-8321) bought 64x Cordrazine for $8.79
tim.fritz@gmail Zydrate REFILL 60 TABLETS 7.19.2004 PAID $25.95 pre-tax
$37.99 Hourai X (20 pills) date:04.15.13 to Mr. Vittorio - jvittori@akamai.net 623-555-0843
5/22/14 - ehatzle@yahoocom pick-up 48 pack Alprazaline - $42.87 USD
15 Progenitorivox at $10.50 refilled for (james77@inbox.com, 8795552304) : 11/14/07

Date	Email	Phone #	Brand	Amt.	Price $
10/3/2009	benbitd@mit.edu	1-470-555-8321	Cordrazine	64	8.79
7.19.2004	tim.fritz@gmail		Zydrate	60	25.95
04.15.13	jvittorio@akamai.net	723-555-0843	Hourai X	20	37.99
5/22/14	ehatzle@yahoocom		Alprazaline	48	42.87
11/14/07	james77@inbox.com	8795552304	Progenitorivox	15	10.50


home
projects
fields
label
connect
review
query
Projects
Create new Project
Project name
Select txt file
Create Project


hadoop
home
projects
fields
label
connect
review
query
Projects
Create new Project
Project name
Select txt file
Create Project
Existing Projects
hadoop
Switch
Delete Project

hadoop

- home
- projects
- fields
- label
- connect
- review
- query

Label training samples

Show 10 entries

Search:

record_id	txt_record	date	time	codepath	src_ip	d
0	2012-01-04 00:01:23,180 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-2281137920769708011_1116 src: /127.0.0.1:32981 dest: /127.0.0.1:50010	date	time	codepath	src_ip	d
1	2012-01-04 00:01:23,184 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32981, dest: /127.0.0.1:50010, bytes: 3758, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_-2281137920769708011_1116, duration: 2016056	date	time	codepath	src_ip	d
2	2012-01-04 00:01:23,185 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_-2281137920769708011_1116 terminating	date	time	codepath	src_ip	d
3	2012-01-04 00:01:23,291 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_3766031435252346505_1117 src: /127.0.0.1:32982 dest: /127.0.0.1:50010	date	time	codepath	src_ip	d
4	2012-01-04 00:01:23,293 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32982, dest: /127.0.0.1:50010, bytes: 265, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID:	date	time	codepath	src_ip	d

txt_record	date	time	codepath	src_ip	dest_ip	block
2012-01-04 00:01:23,180 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-2281137920769708011_1116 src: /127.0.0.1:32981 dest: /127.0.0.1:50010	2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32981	127.0.0.1:50010	2281137920769708011
2012-01-04 00:01:23,184 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32981, dest: /127.0.0.1:50010, bytes: 3758, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_-2281137920769708011_1116, duration: 2016056	2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32981	127.0.0.1:50010	2281137920769708011
2012-01-04 00:01:23,185 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_-2281137920769708011_1116 terminating	2012-01-04	00:01:23	org.apache.hadoop.hdf	src_ip	dest_ip	2281137920769708011
2012-01-04 00:01:23,291 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_3766031435252346505_1117 src: /127.0.0.1:32982 dest: /127.0.0.1:50010	2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32982	127.0.0.1:50010	31435252346505_1117

9	2012-01-04 00:01:23,409 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-965793757262168743_1119 src: /127.0.0.1:32984 dest: /127.0.0.1:50010	date	time	codepath	src_ip

Showing 1 to 10 of 33 entries

Previous 1 2 3 4 Next

Save labels

hadoop

- home
- projects
- fields
- label
- connect
- review
- query

Ambiguities

src_ip, dest_ip

Auxiliary Fields

Field names (csv)

Add Auxiliary Fields

Connections

date → date

Add Connections

Custom Tokens

Regex

Add Custom Tokens

Auxiliary Fields

Field names (csv)

Add Auxiliary Fields

src dest

Connections

date → date

Add Connections

- dest → dest_ip
- src → src_ip

Custom Tokens

Regex

Add Custom Tokens

dest src

date	time	codepath	src_ip	dest_ip	block	src	dest
2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32981	127.0.0.1:50010	2281137920769708011	src	dest
2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32981	127.0.0.1:50010	2281137920769708011	src	dest
2012-01-04	00:01:23	org.apache.hadoop.hdf	src_ip	dest_ip	2281137920769708011	src	dest
2012-01-04	00:01:23	org.apache.hadoop.hdf	127.0.0.1:32982	127.0.0.1:50010	3766031435252346505	src	dest

hadoop

- home
- projects
- fields
- label
- connect
- review
- query

Auxiliary Fields

Field names (csv)

Add Auxiliary Fields

src dest

Connections

date → date

Add Connections

- dest → dest_ip
- src → src_ip

Custom Tokens

Regex

Add Custom Tokens

dest src

hadoop

home

projects

fields

label

connect

review

query

Smoothing: 0 Extract!

Show 10 entries

Search:

record_id	txt_record	confidence	date	time	codepath	src_ip	dest_ip	block	src	dest

No data available in table

Showing 0 to 0 of 0 entries

Previous Next

txt_record	confidence	date	time	codepath	src_ip	dest_ip	bl
2012-01-04 00:01:23,180 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_-2281137920769708011_1116 src: /127.0.0.1:32981 dest: /127.0.0.1:50010	3.73980316945e-50	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	[u'127.0.0.1:32981']	[u'127.0.0.1:50010']	[u
2012-01-04 00:01:23,184 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32981, dest: /127.0.0.1:50010, bytes: 3758, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_-2281137920769708011_1116, duration: 2016056	1.13664076067e-105	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace']	[u'127.0.0.1:32981']	[u'127.0.0.1:50010']	[u
2012-01-04 00:01:23,185 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_-2281137920769708011_1116 terminating	1.84902779147e-35	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	None	None	[u
2012-01-04 00:01:23,291 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving block blk_3766031435252346505_1117 src: /127.0.0.1:32982 dest: /127.0.0.1:50010	4.24080433853e-48	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	[u'127.0.0.1:32982']	[u'127.0.0.1:50010']	[u
2012-01-04 00:01:23,293 INFO org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace: src: /127.0.0.1:32982, dest: /127.0.0.1:50010, bytes: 265, op: HDFS_WRITE, cliID: DFSClient_-603743753, offset: 0, srvID: DS-292194659-127.0.1.1-50010-1324763300176, blockid: blk_3766031435252346505_1117, duration: 552828	1.28891036528e-103	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode.clienttrace']	[u'127.0.0.1:32982']	[u'127.0.0.1:50010']	[u
2012-01-04 00:01:23,293 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: PacketResponder 0 for block blk_3766031435252346505_1117 terminating	2.09673202701e-33	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	None	None	[u
2012-01-04 00:01:23,324 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Receiving	3.73980316945e-50	[u'2012-01-04']	[u'00:01:23']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode']	[u'127.0.0.1:32983']	[u'127.0.0.1:50010']	[u

2012-01-04 00:01:24,862 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling block blk_-2289356614634084790_1114 file /hadoop/dfs/data/current/blk_-2289356614634084790 for deletion	0.0	None	None	None	None	[u'2012-01-04 00:01:24, org.apache.hadoop.hdfs Scheduling block blk_-2 /hadoop/dfs/data/curren for deletion\n']
2012-01-04 00:01:24,862 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling block blk_6186891375771801408_1115 file /hadoop/dfs/data/current/blk_6186891375771801408 for deletion	0.0	None	None	None	None	[u'2012-01-04 00:01:24, org.apache.hadoop.hdfs Scheduling block blk_61 /hadoop/dfs/data/curren deletion\n']
2012-01-04 00:01:24,863 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Deleted block blk_-2289356614634084790_1114 at file /hadoop/dfs/data/current/blk_-2289356614634084790	0.0	None	None	None	None	[u'2012-01-04 00:01:24, org.apache.hadoop.hdfs Deleted block blk_-2289 /hadoop/dfs/data/curren
2012-01-04 00:01:24,863 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Deleted block blk_6186891375771801408_1115 at file /hadoop/dfs/data/current/blk_6186891375771801408	0.0	None	None	None	None	[u'2012-01-04 00:01:24, org.apache.hadoop.hdfs Deleted block blk_6186 /hadoop/dfs/data/curren


Smoothing:
1e-4
Extract!

2012-01-04 00:01:24,862 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling block blk_-2289356614634084790_1114 file /hadoop/dfs/data/current/blk_-2289356614634084790 for deletion	2.43909687879e-72	[u'2012-01-04']	[u'00:01:24']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode', u'/hadoop/dfs/data/current/blk']	None	None
2012-01-04 00:01:24,862 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Scheduling block blk_6186891375771801408_1115 file /hadoop/dfs/data/current/blk_6186891375771801408 for deletion	5.4099535517e-69	[u'2012-01-04']	[u'00:01:24']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode', u'/hadoop/dfs/data/current/blk']	None	None
2012-01-04 00:01:24,863 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Deleted block blk_-2289356614634084790_1114 at file /hadoop/dfs/data/current/blk_-2289356614634084790	4.60989414211e-71	[u'2012-01-04']	[u'00:01:24']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode', u'/hadoop/dfs/data/current/blk']	None	None
2012-01-04 00:01:24,863 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Deleted block blk_6186891375771801408_1115 at file /hadoop/dfs/data/current/blk_6186891375771801408	1.02248145221e-67	[u'2012-01-04']	[u'00:01:24']	[u'org.apache.hadoop.hdfs.server.datanode.DataNode', u'/hadoop/dfs/data/current/blk']	None	None

7. Bibliography

[1] Box, G. E., & Draper, N. R. (1987). Empirical model-building and response surfaces. New York: Wiley.

[2] Data Wrangler. (n.d.). Retrieved January 28, 2016, from http://vis.stanford.edu/wrangler/

[3] Entity Extractor. (2013). Retrieved January 28, 2016, from http://www.basistech.com/text-analytics/rosette/entity-extractor/

[4] Hidden Markov Models in Python. (n.d.). Retrieved January 28, 2016, from http://www.cs.colostate.edu/~anderson/cs440/index.html/doku.php?id=notes:hmm2

[5] Kandel, S., Paepcke, A., Hellerstein, J., & Heer, J. (2011). Wrangler: Interactive Visual Specification of Data Transformation Scripts. Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems - CHI '11.

[6] Le, V., & Gulwani, S. (2014). FlashExtract: A Framework for Data Extraction by Examples. ACM SIGPLAN Notices SIGPLAN Not., 49(6), 542-553.

[7] Marr, D. (1977). Artificial intelligence—A personal view. Artificial Intelligence, 9(1), 37-48.

[8] Mayer, M., Soares, G., Grechkin, M., Le, V., Marron, M., Polozov, O., . . . Gulwani, S. (2015). User Interaction Models for Disambiguation in Programming by Example. Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15.

[9] PROSE Playground. (n.d.). Retrieved January 28, 2016, from https://prose-playground.cloudapp.net/

[10] Rabiner, L. R. (1990). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Readings in Speech Recognition, 267-296.

[11] Sarkar, A. (2013, February 28). HMM6-supervised-learning.pdf [PDF]. Burnaby, British Columbia: Simon Fraser University. Available at http://www.cs.sfu.ca/~anoop/teaching/CMPT-413-Spring-2014/HMM6-supervised-learning.pdf

[12] Siddiqi, S. M., Gordon, G. J., & Moore, A. W. (2007). Fast State Discovery for HMM Model Selection and Learning. Proceedings of the International Conference on Artificial Intelligence and Statistics, 492-499. Retrieved from https://www.cs.cmu.edu/~ggordon/siddiqi-gordon-moore.fast-hmm.pdf

[13] Tamr's Take ... On Human-Machine Collaboration -- Tamr, Inc. (2015, April 09). Retrieved January 28, 2016, from http://www.tamr.com/tamrs-take-on-human-machine-collaboration/

[14] Trim, C. (2013, January 23). Mark as Duplicate. Retrieved January 28, 2016, from https://www.ibm.com/developerworks/community/blogs/nlp/entry/tokenization?lang=en

[15] Viterbi algorithm. (n.d.). Retrieved January 28, 2016, from https://en.wikipedia.org/wiki/Viterbi_algorithm

[16] Winston, P. H. (1970). Learning Structural Descriptions from Examples (PhD. thesis, Massachusetts Institute of Technology, 1970). Cambridge: Massachusetts Institute of Technology.

[17] Winston, P. H. (1984). Artificial intelligence. Reading, MA: Addison-Wesley.