Anchor Finding Interface V1.0

Yoni Halpern

November 17, 2014

Hi! This document accompanies the initial release of the anchor interface described in: "Using Anchors to Estimate Clinical State without Labeled Data" by Y. Halpern, Y.D. Choi, S. Horng, D. Sontag. To appear in the American Medical Informatics Association (AMIA) Annual Symposium, Nov. 2014.

1 Contact Information

Please direct questions to Yoni Halpern: halpern@cs.nyu.edu

2 Quick Start

2.1 Installation

Clone the github repository for AnchorExplorer with the following command:

```
$ git clone https://github.com/yhalpern/anchorExplorer.git
$ cd anchorExplorer
```

We have tested the interface on a system with the following properties:

- Mac or unix OS
- python 2.6 or 2.7
- numpy 1.8.1 and scipy 0.13.3
- scikit-learn 0.15
- networkx 1.8.1
- Tkinter Revision 81008
- ttk 0.3.1

Required python packages are listed a file, requirements.txt and can be installed using pip:

```
$ pip install -r requirements.txt
```

If you have trouble with Tkinter and ttk, try installing the ActiveState community edition of Python `http://www.activestate.com/activepython/`.

2.2 Data

To start, you need to have patient records in xml format. However, if you want to test that you can get things up and running with dummy data you can use the following command to generate 1000 properly formatted "random" patient records and store them in patients.xml. A second, more involved example can be found in Section 4.3.

```
$ python generate_patients.py 1000 > patients.xml
```

2.3 Settings

An example settings file is provided in the examples/ directory. To customize this file for your own data, see section 4.3.

2.4 Preprocess Data

Use the preprocess_patients.py script to preprocess the data in patients.xml for easy lookups and storage. The interface will read from the files generated by this script. Using the example settings file:

```
$ python preprocess_patients.py 1000 patients.xml examples/settings.xml
```

This script does some simple negation detection, bigram detection and stopword removal. If you wish to use your own custom language processing pipeline, see section 4.2.

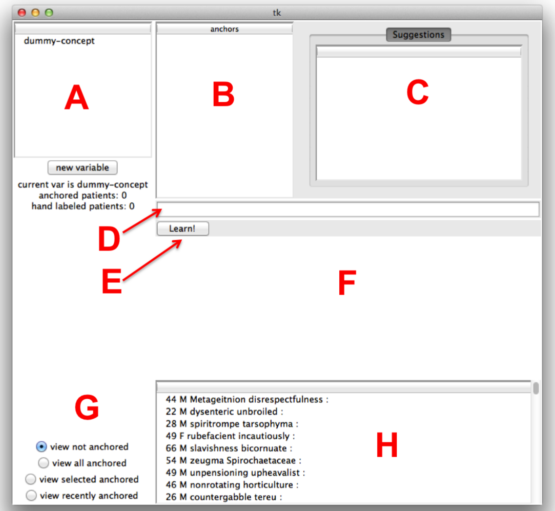
2.5 Create an anchors directory

```
$ mkdir anchors
```

2.6 Run the interface

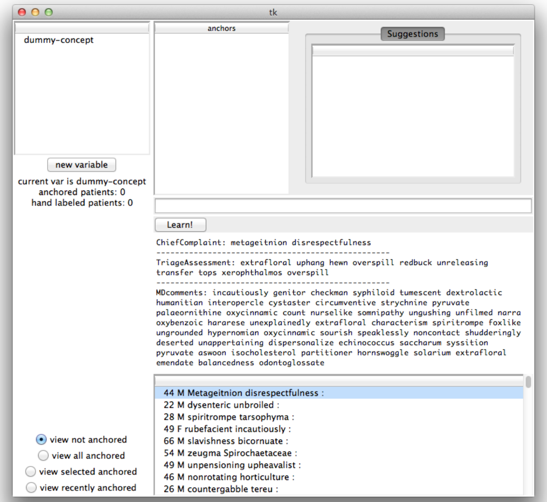
To run the interface using the example settings file:

```
$ python gui.py examples/settings.xml
```



current var is dummy-concept
anchored patients: 0
hand labeled patients: 0

```
44 M Metageitnion disrespectfulness :
22 M dysenteric unbroiled :
28 M spiritrompe tarsophyma :
49 F rubefacient incautiously :
66 M slavishness bicornuate :
54 M zeugma Spirochaetaceae :
49 M unpensioning upheavalist :
46 M nonrotating horticulture :
26 M countergabble tereu :
```



current var is dummy-concept
anchored patients: 0
hand labeled patients: 0

```
ChiefComplaint: metageitnion disrespectfulness
------------------------------------------------
TriageAssessment: extrafloral uphang hewn overspill redbuck unreleasing
transfer tops xerophthalmos overspill
------------------------------------------------
MDcomments: incautiously genitor checkman syphiloid tumescent dextrolactic
humanitian interopercle cystaster circumventive strychnine pyruvate
palaeornithine oxycinnamic count nurselike somnipathy ungushing unfilmed narra
oxybenzoic hararese unexplainedly extrafloral characterism spiritrompe foxlike
ungrounded hypernomian oxycinnamic sourish speaklessly noncontact shudderingly
deserted unappertaining dispersonalize echinococcus saccharum syssition
pyruvate aswoon isocholesterol partitioner hornswoggle solarium extrafloral
emendate balancedness odontoglossate
```

```
44 M Metageitnion disrespectfulness :
22 M dysenteric unbroiled :
28 M spiritrompe tarsophyma :
49 F rubefacient incautiously :
66 M slavishness bicornuate :
54 M zeugma Spirochaetaceae :
49 M unpensioning upheavalist :
46 M nonrotating horticulture :
26 M countergabble tereu :
```

Figure 1: (Left) Screenshot when the interface is first activated. Panes are annotated in red and each one is explained in detail in Section 3. (Right) Viewing a patient in more detail. Random words are inserted for patient text here.

2.7 Exploring the data

You can explore the data on the patient visit level with the patient summary display[1] (H) and detailed display (F). Selecting a patient in the summary display shows the patient's record in more detail above it. In this simple example, we are only showing three text fields, but more data can be shown here.

2.8 Create a new cohort

Pushing the "new variable" button (below A) will pop up a dialog to create a new cohort variable. You will be prompted to provide a name.

2.9 Adding an anchor

Choose a word that appears in the patient data (e.g. for the randomly generated data, try "bioluminescence") and enter it in the text input box (D).

[1]This display sometimes gets initialized with a height of zero at the bottom of the window and may need to be dragged up in order to be visible.

2.10 Filtering the data

Choose to filter for patients that have the anchor or view only patients that do not have the anchor by using the radio buttons to change the filters (G).

2.11 Learning a model

Until now, you have just been exploring the patient data and applying filters, but no learning has taken place. Now try pushing the "Learn" button (E). After a few seconds, your patients should now be ranked in order of likelihood of belonging to the cohort and suggestions for additional anchors will appear in the suggestions pane (C).

2.12 Saving state

Try closing the window and opening it up again. Your cohorts and anchors should be preserved.

3 Anatomy of the interface

3.1 Cohort Selection Menu (A)

Allows you to select cohorts, and create new ones. Some commands:

- New variable button to create a new cohort
- '-' delete cohort
- 'r' rename cohort

3.2 Anchor Display (B)

Displays the current anchors for the selected topic. Commands:

- Enter text into text box and push enter to add a new anchor
- '-' delete selected anchor
- Anchors can also be added using the anchor suggestion pane (Top Right)

3.3 Anchor Suggestion Pane (C)

Displays suggestions for anchors and allows for navigation and adding of structured anchors. Commands:

- "+" to add selected anchor suggestion

Important: The anchor suggestions are not necessarily all good - choosing anchors requires judgment of whether to accept or reject the automated suggestions. They are meant to trigger associations, not tell you what makes a good anchor or not!

3.4 Anchor Input Box (D)

Type anchors in here, push enter to apply them. You will need to push learn to relearn the model after adding an anchor.

3.5 Learn Button (E)

Learns a model using the currently selected anchors.

3.6 Detailed Patient Display (F)

Detailed display of the patient selected in Patient List (Bottom Right). Anchors are highlighted red. Structured data types can be clicked to navigate directly to the relevant part of the hierarchy (displayed in Anchor Suggestion Pane).

3.7 Patient Filters (G)

Possible options here:

- view not anchored: patients that do not have an anchor.
- view all anchored: all patients that have an anchor.
- view selected anchor: view patients that have the selected anchor (selected in pane B).
- view recent anchor: view patients added by the most recent anchor.

3.8 Patient List (H)

List of patients with summary information. These patients can be filtered using the radio buttons on the left (e.g. view all anchored, view not anchored, etc). After the concept has been “learned” using the Learn Button, this list will be ranked in order of how highly the patient fits the currently selected concept.

Usage:

- arrow keys to navigate
- “+” mark patient as a positive example
- “-” mark patient as a negative example
- “0” remove patient markings

4 Customizations

This section is still incomplete, more information coming here shortly. Please contact if you have questions about how to use this tool on your data.

4.1 patientSets

This section describes which patients should be selected for visualization. If you have many patients, the interface may run slowly, here you can show that you only want to use the first 10,000 patients for an initial exploration.

```
<patientSets>
    <set name='train' start='0' end='10000'/>
    <set name='validate' start='0' end='15000'/>
</patientSets>
```

The only important patientSet in this version of the code are the “train” and “validate” set. The start and end fields indicate indices of patients as they are listed in the file visitIDs which is generated by the preprocessing script. Only patients from the validate set are shown. Patients in the train set are not shown.

4.2 Language Parsing

Coming soon... more information on customizing the language parser.

4.3 Data Fields

Use the settings.xml file to customize the interface for your particular dataset. An example settings.xml file is provided along with the interface code in the examples directory. Most of it can be left as is. The most important parts to customize are the `dataTypes` and `displaySettings` sections.

4.3.1 Data Types

We divide data into different *types* (e.g. text, age, sex, medications, diagnoses, procedures, etc.). Each type is denoted by a `datum` tag and contains a number of `field` tags that describe where this data appears in a patient xml representation.

For example, here is a sample patient representation in xml format generated by the random patient generator:

```
//patient.xml
<visit>
  <index>qVwLYjLKqlkZhvkf</index>
  <MDcomments> SOME TEXT</MDcomments>
  <Age> 44 </Age>
  <Sex>M</Sex>
  <ChiefComplaint> SOME TEXT</ChiefComplaint>
  <TriageAssessment>SOME TEXT</TriageAssessment>
</visit>
```

Important: Every patient instance must be enclosed in a `visit` tag and must have a unique `index` tag. All other fields are customizable. Here is a section of settings.xml that could be used for patient records with this schema.

```
//settings.xml
1. <datum type=‘text’ heirarchy=‘’ prefix=‘’>
2.     <field name=‘MDcomments’ path=‘.’/>
3.     <field name=‘ChiefComplaint’ path=‘.’/>
4.     <field name=‘TriageAssessment’ path=‘.’/>
5. </datum>
```

Line 1 of settings.xml says that there is a type of data called **text**. That it is not hierarchical and that it should not be represented with any special prefix.

Lines 2-4 show where in the patient records it can be found (ie. the MDcomments, ChiefComplaint and TriageAssessment sections). The final representation of the **text** data will be a concatenation of these three fields.

4.3.2 Display Settings

For each type of data described in Section 4.3.1, you may want to customize where it is displayed (or whether it is displayed at all). Here you can specify what appears in the `patientSummary` section and the `detailedDisplay`.

The following snippet says that Age, Sex and ChiefComplaint should be displayed as the patient summary:

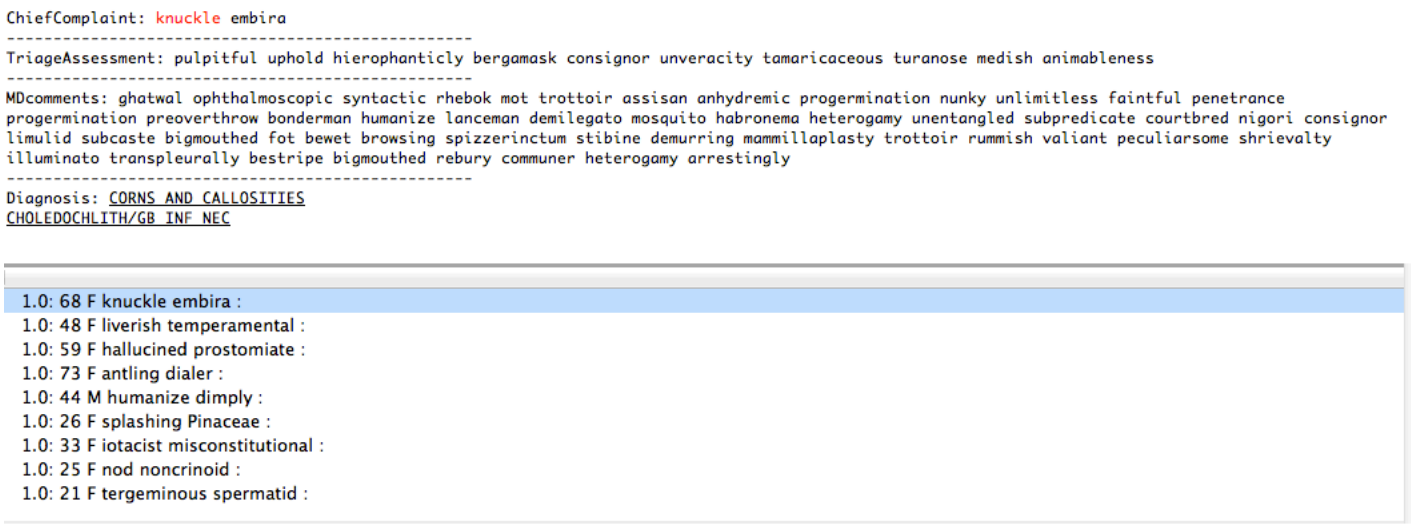
```
<patientSummary>
    <displayFields>
        <field name=’Age’/>
        <field name=’Sex’/>
        <field name=’ChiefComplaint’/>
    </displayFields>
</patientSummary>
```

And the following snippet says that ChiefComplaint, TriageAssessment, MDcomments should be displayed in the detailed patient description:

```
<detailedDisplay>
    <displayFields>
```

```
        <field name='ChiefComplaint' path='.'/>
        <field name='TriageAssessment' path='.'/>
        <field name='MDcomments' path='.'/>
    </displayFields>
</detailedDisplay>
```

The resulting display is shown in Figure 2.



```
ChiefComplaint: knuckle embira
--------------------------------------------------
TriageAssessment: pulpitful uphold hierophanticly bergamask consignor unveracity tamaricaceous turanose medish animableness
--------------------------------------------------
MDcomments: ghatwal ophthalmoscopic syntactic rhebok mot trottoir assisan anhydremic progermination nunky unlimitless faintful penetrance
progermination preoverthrow bonderman humanize lanceman demilegato mosquito habronema heterogamy unentangled subpredicate courtbred nigori consignor
limulid subcaste bigmouthed fot bewet browsing spizzerinctum stibine demurring mammillaplasty trottoir rummish valiant peculiarsome shrievalty
illuminato transpleurally bestripe bigmouthed rebury communer heterogamy arrestingly
--------------------------------------------------
Diagnosis: CORNS AND CALLOSITIES
CHOLEDOCHLITH/GB INF NEC
```

```
1.0: 68 F knuckle embira :
1.0: 48 F liverish temperamental :
1.0: 59 F hallucined prostomiate :
1.0: 73 F antling dialer :
1.0: 44 M humanize dimply :
1.0: 26 F splashing Pinaceae :
1.0: 33 F iotacist misconstitutional :
1.0: 25 F nod noncrinoid :
1.0: 21 F tergeminous spermatid :
```

Figure 2: Customizing the patient display as described in section 4.3.2. The bottom list view shows Age, Sex and Chief Complaint for every patient. The top detailed view of a single patient shows fields: ChiefComplaint, TriageAssessment,MDcomments. We display random words instead of real patient notes.

4.4 Structured Data Types

Some data elements can be described as belonging to a hierarchy of items, (e.g. ICD9 codes, NDC codes, etc). We support exploring data hierarchies and adding anchors from hierarchically typed data. In the following section, we demo this capability in order to view diagnosis codes in MIMIC data alongside text.

4.5 Customization demo

This demo will walk you through how to include a new data field, in this case ICD9 codes to be viewed in the user interface. It assumes that you have access to MIMIC-II data.
First download the flat note files from `http://physionet.org/mimic2/flat_files/`.

4.5.1 Preparing data

The commands from this demo can be found in a script mimic_demo.sh which takes a single argument which is the path to the mimic-II flat files.

As before, we begin by formatting the patient data in a patients.xml file.

```
$ python get_mimic_data.py path/to/mimic_files/00 examples/mimic_fields.txt
> patients.xml
```

Looking at the patients.xml file, you will see that ICD9 codes are recorded in the dataset.

We would like to create a *data dictionary* to map these codes to plaintext and a *data structure* to store these codes in a hierarchical manner.

First we need two tab-separated files that will store this data: X.names and X.edges. In general we assume that the user supplies these files for any structured data type of interest. For this example, we provide a script examples/ICD9/load_ICD9_structure.py which downloads the ICD9 hierarchy from a public git repository and creates these formatted files. **This hierarchy is incomplete and does not contain entries for every ICD9 code in MIMIC-II, but it serves as a good illustration**.

```
$ cd examples/ICD9
$ python load_ICD9_structure.py
```

We now want to build these into the structured format that the interface reads from.

```
$ cd ../..
$ python build_structured_rep.py code examples/ICD9/code
$ ls Structures
codeDict.pk    codeStruct.pk
```

Now we need to update settings.xml to recognize this new structured data type. An example settings file is found in examples/ICD9/settings.xml. Below is an excerpt of the relevant lines:

```
#lines 28-30
 <datum type='code' heirarchy='Structures/codeStruct.pk' prefix='code_'
  dictionary='Structures/codeDict.pk' realtime='true'>
    <field name='Diagnosis' path='D_code' display='D_name'/>
 </datum>


 #lines 41-45
 <displayFields>
    <field name='ChiefComplaint' path='.'/>
    <field name='TriageAssessment' path='.'/>
    <field name='MDcomments' path='.'/>
    <field name='Diagnosis' path='.'/>
 </displayFields>
```

We can now run the preprocessing script to import the data with the structured ICD9 codes included.

```
$ python preprocess_patients.py 1000 patients.xml examples/ICD9/settings.xml
```

And run the interface:

```
$ python gui.py examples/ICD9/settings.xml
```

4.5.2 Using the interface

Try to create a new cohort and add an anchor: As an illustrative example, we will build the cohort of recent births.

Create a new cohort:
Push the **new variable** button (Pane A in Figure 2) and enter "recent birth".

Adding a bad anchor:
We'll start with a straw-man bad anchor for recent birth. Say we hypothesize that the word "mother" is a good anchor for recent birth (this would mean that if mother is mentioned in the note, it is almost certainly a recent birth. This is almost certainly not true, but let's run with it and see what happens.)

Type "mother" in the anchor entry box and push enter (Pane D in Figure 2).

Detecting the bad anchor:
Using the radio buttons on the bottom left, select **view all anchors**. Scrolling through the patients, you will see the anchors highlighted in red. Scrolling through the first few patients should reveal that "mother" occurs in many contexts other than recent birth, including family history, contact information etc.

Remove the bad anchor and propose a new one:
We can remove the bad anchor by selecting it and pushing '-'. Now let's add "neonatology" as an anchor. This is a better anchor because it is highly specific. Once again, type it into the anchor entry box, push enter. You can scroll through patients again to confirm that this is a better anchor.

Now next to the suggestions pane there should be an option 'code' which will display ICD9 codes in a structured version that can be added to the anchor list with the '+' key. Adding a parent in the hierarchy will add all of the children.

5 More Information

Contact: Yoni Halpern - halpern@cs.nyu.edu