MASK Framework

Nikola Milosevic

Jun 04, 2019

CONTENTS:

CHAPTER

ONE

INTORDUCTION

MASK Framework is an open-source framework for de-identification of medical free-text data

In this project, we will develop an open-source framework for automated de-identification of medical textual data. Such data contains information that can be utilized to support clinical research, but its native form contains sensitive personal identifiable information (PII) that should not be accessed by anyone who does not provide direct clinical care.

The project aims to enhance the current processes and build an open-source platform that can be used for flexible masking of personal information, ensuring that de-identified medical text still contains enough information to facilitate research.

In order to facilitate flexibility, the de-identification system has to be configurable by the user in terms of:

- Types of PII that have to be identified in free-text data;
- Approaches to masking of the identified data (keep, redact, map, etc.);
- Disclosure risk analysis that is performed on the data;
- The methodology that is applied for each of the steps.

CHAPTER

TWO

ARCHITECTURAL CONSIDERATIONS

2.1 Configuration

The requirements for the configuration file are:

- Store the information about algorithms that should be used for NER
- This can be done for per entity
- **Store information about masking**
  - Which named entities to mask
  - How these named entities should be masked
  - There can be a choice: do not mask, map and redact
- Talk to ICES what should we implement as examples (name, postcode, age intervals)
- User can pick algorithm for mapping
- Algorithms for mapping can be added as plugins
- Mapping algorithms should be defined for each NER

2.2 Architectural consideration for extendable framework and configuration

For named entity recognition algorithms there are following considerations:

- All implementations should be implemented in a single file as a class
- All implementations should be stored in a single folder
- All implementations should inherit same abstract class, implement method initialize (should load the models), perform_NER (takes string and returns an array of tuples with class, begin span, end span).
- They should all return a subset of defined classes (PATIENT_NAME, DOCTOR_NAME, PROFESSION, ADDRESS, CITY, COUNTRY, POST_CODE, PHONE_NUMBER, EMAIL, WEB_ADDRESS, PATIENT_ID, DOCTOR_ID, ORGANIZATION, DATE)
- Defined functions in the config file should correspond to the class and file names in this directory

For extensions related to masking functions there are following considerations:

- All implementations should be implemented in a single file as a class
- All implementations should be all stored in a single folder

- All implementations should inherit the same abstract class and implement "mask" method that takes as input string to be masked and return masked string (either mapped or redacted in a particular manner).
- Defined functions in the config file should correspond to the class and file names in this directory

2.3 Configuration file example and explanation

Example of configuration file:

```
<project>
    <project_name>Masking v1</project_name>
    <project_start_date>30/05/2019</project_start_date>
    <project_owner>Nikola Milosevic</project_owner>
    <project_owner_contact>nikola.milosevic@manchester.ac.uk</project_owner_contact>
    <algorithms>
        <entity>
            <entity_name>NAME</entity_name>
            <original_name>NAME</original_name>
            <algorithm>NER_CRF</algorithm>
            <masking_type>Redact</masking_type>
        </entity>
        <entity>
            <entity_name>DATE</entity_name>
            <original_name>DATE</original_name>
            <algorithm>NER_CRF</algorithm>
            <masking_type>Mask</masking_type>
            <masking_class>Mask_date</masking_class>
        </entity>
    </algorithms>
    <dataset>
        <dataset_location>dataset/input</dataset_location>
        <data_output>dataset/output</data_output>
    </dataset>
</project>
```

2.4 Explanation

The whole configuration is wrapped in <project> tag. The user can name the project (using <project_name>), and give some basic information about creator and contact details. For each entity, user would like to mask, he/she needs to create <entity> tag.

Inside <entity> tag, user has to define entity name (using entity_name tag), he can specify original name that his named entity recognizer outputs (using original_name tag), specify NER algorithm for recognition (using <algorithm> tag) and define masking. Masking can be defined by specifying masking type (using masking_type tag). Possible values for masking type are:

- Nothing - does nothing, does not redact or mask entity, but leaves it in text.
- Mask - masks entity with another string. The way of masking has to be defined with the masking_class tag.
- Redact - redacts the entity (setting either XXX or entity name - to be discussed in the future).

CHAPTER

THREE

CLASSES AND FUNCTIONS

mask_framework.py – Main MASK Framework module

class mask_framework.**Configuration**(*configuration='configuration.cnf'*)
Class for reading configuration file

Init function that can take configuration file, or it uses default location: configuration.cnf file in folder where mask_framework is

mask_framework.**main**()
Main MASK Framework function

ner_plugins - a set of modules that can perform named entity recognition. Basically, plugins for different kinds of named entity recognition

class mask_framework.**Configuration**(*configuration='configuration.cnf'*)
Class for reading configuration file

Init function that can take configuration file, or it uses default location: configuration.cnf file in folder where mask_framework is

class ner_plugins.NER_CRF.**NER_CRF**
The class for executing CRF labelling based on i2b2 dataset (2014).

custom_span_tokenize(*text*, *language='english'*, *preserve_line=True*)
Returns a spans of tokens in text.

Parameters

- **text** – text to split into words
- **language** (*str*) – the model name in the Punkt corpus
- **preserve_line** – An option to keep the preserve the sentence and not sentence tokenize it.

custom_word_tokenize(*text*, *language='english'*, *preserve_line=True*)
Return a tokenized copy of *text*, using NLTK's recommended word tokenizer (currently an improved TreebankWordTokenizer along with PunktSentenceTokenizer for the specified language).

Parameters

- **text** – text to split into words
- **text** – str
- **language** (*str*) – the model name in the Punkt corpus
- **preserve_line** – An option to keep the preserve the sentence and not sentence tokenize it.

doc2features(*sent*)
Transforms a sentence to a sequence of features

Parameters **sent** – a set of tokens that will be transformed to features

perform_NER(*text*)
Implemented function that performs named entity recognition using CRF. Returns a sequence of tuples (token,label).

Parameters **text** – text over which should be performed named entity recognition

tokenize_fa(*documents*)
Tokenization function. Returns list of sequences

Parameters **documents** – list of texts

word2features(*sent*, *i*)
Transforms words into features that are fed into CRF model

Parameters

- **sent** – a list of tokens in a single sentence
- **i** (*int*) – position of a transformed word in a given sentence (token sequence)

class ner_plugins.NER_abstract.**NER_abstract**
Abstract class that other NER plugins should implement

perform_NER(*text*)
Implementation of the method that should perform named entity recognition

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX