ENTITY LINKING IN FOR FUN (CLOJURE)

@Sojoner
AGENDA

★ Motivation
★ Entity linking
★ Reflector overview
★ Represent the data
★ A bit code
★ Learnings
MOTIVATION

- practicing Clojure backend & frontend
- using core.async
- learning about FP
- Gödel-Escher-Bach like information revealer

- How will the Wikipedia reflect a given text when thrown against it?
is the task of determining the identity of entities mentioned in text [...] Wikipedia

You can improve search experience by enriching documents with entities e.g. facet search for persons locations a.o
ENTITY LINKING
SYSTEMS & CHALLENGES

From TagME to WAT: a new Entity Annotator

Francesco Piccinno and Paolo Ferragina
Dipartimento di Informatica
University of Pisa
(piccinno, ferragina)@di.unipi.it

Tulip: Lightweight Entity Recognition and Disambiguation Using Wikipedia-Based Topic Centroids

Marek Lipczak
Faculty of Computer Science
Dalhousie University
Halifax, Canada
lipczak@cs.dal.ca

Arash Koushkestanii
Faculty of Computer Science
Dalhousie University
Halifax, Canada
arash.koushkestanii@dal.ca

Evangelos Milios
Faculty of Computer Science
Dalhousie University
Halifax, Canada
eem@cs.dal.ca

ABSTRACT
In this paper we propose a new entity linking system that is able to automatically annotate entities in a document. The system is based on the idea of using Wikipedia as a knowledge base. The system is called Tulip and it has been evaluated on the TAC KBP 2014 and 2015 datasets.

ABSTRACT
This article presents an ERD system Tulip, a submission to the TAC 2014: Entity Recognition and Disambiguation Challenge. The objective of the proposed system is to spot mentions of entities in a document and link the mentions to corresponding Wikipedia articles. To achieve this, Tulip prunes the set of entity candidates generated by a core module of related entities. The relationship strength is measured as a similarity to a topic centroid generated from entity features. Each entity is represented by an accurate and compact feature vector extracted from a category graph built on information from 120 language versions of Wikipedia. Given the core set of accepted entities, Tulip uses the Wikipedia-based feature vectors to extract more related entities from the document text. The challenge results: first prize in the long document track with F1 score of 0.74 confirms the effectiveness of our system. At the same time, the system was faster than all other submissions with latency under 0.02 seconds.

Categories and Subject Descriptors
1.2.7 [Natural Language Processing]: Text analysis

Simple stuff ;-)
REFLECTOR OVERVIEW & TECHNOLOGIES

; the force
[org.clojure/clojure "1.6.0"]
[com.stuartsierra/component "0.2.2"]
[org.clojure/core.match "0.2.1"]
[org.clojure/clojurescript "0.0-2322"]
[org.clojure/core.async "0.1.346.0-17112a-alpha"]
; graph
[aysylu/loom "0.5.0"]
; IR
[clojure-opennlp "0.3.3"]
[clojurewerkz/elastisch "2.1.0"]
; Frontend
[jom "0.7.3"]
[racehub/om-bootstrap "0.3.2"]
[prismatic/om-tools "0.3.6"]
;; Websockets
[com.taoonsso/sente "1.2.0" :exclusions [org.clojure/clojure]]
;; Backend
[compojure "1.1.9"]
[http-kit "2.1.19"]
[clj-http "1.0.1"]
[ring "1.3.1"]
[ring/ring-defaults "0.1.1"]
;; data on the wire
[com.cognitect/transit-clj "0.8.247"]
[com.cognitect/transit-cljs "0.8.188"]
; logging
[org.clojure/tools.logging "0.3.0"]
[ch.qos.logback/logback-classic "1.1.1"]
<table>
<thead>
<tr>
<th>PERSONS</th>
<th>LOCATIONS</th>
<th>ORGANISATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOHANN WOLFGANG VON GOETHE</td>
<td>LÜTTICH</td>
<td>SOFTM</td>
</tr>
<tr>
<td>LEOPOLD I. (BELGIEN)</td>
<td>PARIS</td>
<td>COMARCH SOFTWARE UND BERATUNG</td>
</tr>
<tr>
<td>LEOPOLD II. (BELGIEN)</td>
<td>BRÜSSEL</td>
<td>KBC BANK DEUTSCHLAND</td>
</tr>
<tr>
<td>LEOPOLD III. (BELGIEN)</td>
<td>PRAG</td>
<td>ERGO VERSICHERUNG</td>
</tr>
<tr>
<td>ALBERT I. (BELGIEN)</td>
<td>GENT</td>
<td>MUSÉE DES TRANSPORTS EN COMMUN DU PAYS DE LIÈGE</td>
</tr>
<tr>
<td>FRANZ JOSEPH I.</td>
<td>MAASTRICHT</td>
<td>BURGBAD</td>
</tr>
<tr>
<td>MAXIMILIAN VON MONTGOLAS</td>
<td>HUY (BELGIEN)</td>
<td>KAMPA (UNTERNEHMEN)</td>
</tr>
<tr>
<td>ALBERT II. (BELGIEN)</td>
<td>KERKRADE</td>
<td>GROHE</td>
</tr>
</tbody>
</table>

**UNDIRECTED WIKIPEDIA PATHS**

- Belgien
- Belgien_nach_Gemeinde
- Lüttich
- Gegenöffentlichkeit
- Kommunikationswissenschaft
- Association_for_Education_in_Journalism_and_Mass_Communication
- Internationale_Organisation
- International_Committee_of_Military_Medicine
- Lüttich
- Besitz
- Volkswirtschaftslehre
- Entwicklungskonomik
- Desretc
- Internationale_Organisation
- International_Committee_of_Military_Medicine
- Lüttich
- Blasonierung
- Heraldik
- Court_of_the_Lord_Lyon
- Edinburgh
- Europa_nach_Ort
- Lüttich
REFLECTOR OVERVIEW & SYSTEM

titles
categories
articles
storage
back
front
"source": {
  "title": "Tiefurt",
  "text": "Tiefurt ist ein Ortsteil der Thüringen",
  "wiki_text": "{{{Infobox Ortsteil einer Gemeinde
  "redirect": false,
  "redirect_page": null,
  "special": false,
  "stub": false,
  "disambiguation": false,
  "category": [
      "Stadtteil von Weimar",
      "Ehemalige Gemeinde (Weimar)",
      "Ort an der Ilm (Saale)",
      "Straßendorf"
    ],
  "link": [
    "Thüringen",
    "Weimar",
    ...
  ],
  "redirects": [],
  "disambiguations": [],
  "incoming_links": [
    "Ingeborg Stein",
    "Liste der Stadtteile von Weimar",
    "Georg Bleyer",
    ...
  ]
}
REPRESENT THE DATA & DATA EXAMPLES

AMBIGUOUS

```
"docs": [
{
"id": "Opposition (Politik)",
"text": [
"Oppositionssprecherin",
"Oppositionsjahre",
"Oppositionskreisen",
"Oppositionskraft",
"oppositionellen Haltung",
"Oppositionssprecher",
"Oppositionen",
"politischer Gegner",
"Oppositionszeit",
"Oppositionspolitiker",
"Oppositionorganisation",
"Fundamentalopposition",
"systemoppositionellen",
"politische Gegner",
"oppositionell",
]
}
]
```

UNIQUE

```
"response": {
"numFound": 1,
"start": 0,
"docs": [
{
"id": "Opposition (Politik)",
"text": [
"Opposition (Politik)
],
"_version_": 1490128805088460800
}
]
```
main starts components

http is the external interface

communicator sends massages

linker does entity linking with solr

search searches pages in ES

wikigraph computes paths

switch glues the core.async channels together
(defn get-system [conf]
  "Create system by wiring individual components so that component/start
  will bring up the individual components in the correct order."
  (component/system-map
    :communicator-channels (comm/new-communicator-channels)
    :communicator (component/using (comm/new-communicator) {:channels :communicator-channels})
    :linker-channels (linker/new-linker-channels)
    :linker (component/using (linker/new-linker-conf) {:channels :linker-channels})
    :search-channels (search/new-search-channels)
    :search (component/using (search/new-search-conf) {:channels :search-channels})
    :graph-channels (graph/new-graph-channels)
    :graph (component/using (graph/new-graph-conf) {:channels :graph-channels})
    :http (component/using (http/new-http-server-conf) {:communicator :communicator})
    :switchboard (component/using (sw/new-switch) {:comm-chans :communicator-channels
      :search-chans :search-channels
      :linker-chans :linker-channels
      :graph-chans :graph-channels}))
)

(def system (get-system conf))

(defn -main [& args]
  (log/info "Application starting")
  (alter-var-root #'system component/start)
  (log/info "Application started"))
(ns back.knubr.switch
  (:gen-class)
  (:require
    [clojure.tools.logging :as log]
    [com.stuartsierra.component :as component]
    [clojure.core.async :as async :refer [chan mult tap pipe]]))

(defrecord Switch [comm-chans search-chans linker-chans graph-chans]
  component/Lifecycle
  (start [component] (log/info "Starting Switchboard Component")
    (pipe (:query comm-chans) (:query search-chans))
    (pipe (:entity-details comm-chans) (:entity-details search-chans))
    (pipe (:graph comm-chans) (:graph graph-chans))
    (pipe (:graph-results graph-chans) (:graph-results comm-chans))
    (pipe (:linker comm-chans) (:linker linker-chans))
    (pipe (:linker-results linker-chans) (:linker-results comm-chans))
    (pipe (:query-results search-chans) (:query-results comm-chans))
    (pipe (:entity-details-results search-chans) (:entity-details-results comm-chans)))
  (stop [component] (log/info "Stop Switchboard Component")))

(defn new-switch [] (map->Switch {}))
(def ring-defaults-config (assoc-in ring.middleware.defaults/site-defaults [:security :anti-forgery]
    {:read-token (fn [req] (-> req :params :csrf-token))})))
(defn- static-html [file-name] (content-type (resource-response file-name {:root "public"})) "text/html"
(defrecord Httpserver [conf communicator server]
  component/Lifecycle
  (start [component] (log/info "Starting HTTP Component")
    (defroutes my-routes ; created during start so that the correct communicator instance is used
      (GET "/" [] (static-html "index.html"))
      (GET "/chsk" req ((:ajax-get-or-ws-handshake-fn communicator) req))
      (POST "/chsk" req ((:ajax-post-fn communicator) req))
      (route/resourses "/") ; Static files, notably public/main.js (our cljs target)
      (route/not-found "Page not found"))
    (let [my-ring-handler (ring.middleware.defaults/wrap-defaults my-routes ring-defaults-config)
      server (http-kit-server/run-server my-ring-handler {:port (:port conf)})
      uri (format "http://localhost:%s/" (:local-port (meta server)))
      (log/info "Http-kit server is running at" uri)
      (assoc component :server server))
    (stop [component] (log/info "Stopping HTTP Server")
      (server :timeout 100)
      (assoc component :server nil)))
(defn new-http-server [conf] (map->Httpserver {:conf conf}))
A BIT CODE

(def ring-defaults-config (assoc-in ring.middleware.defaults/site-defaults [:security :anti-forgery] 
  {:read-token (fn [req] (-> req :params :csrf-token)))})

(defn- static-html [file-name] (content-type (resource-response file-name {:root "public"})) "text/html"

(defrecord Httpserver [conf communicator server]
  component/Lifecycle
  (start [component] (log/info "Starting HTTP Component")
    (defroutes my-routes ; created during start so that the correct communicator instance is used
      (GET "/" []) (static-html "index.html")
      (GET "/chsk" req ((:ajax-get-or-ws-handshake-fn communicator) req))
      (POST "/chsk" req ((:ajax-post-fn communicator) req))
      (route/resources "/") ; Static files, notably public/main.js (our cljs target)
      (route/not-found "Page not found"))
    (let [my-ring-handler (ring.middleware.defaults/wrap-defaults my-routes ring-defaults-config)
      server (http-kit-server/run-server my-ring-handler {:port (:port conf)})
      uri (format "http://localhost:%s/" (:local-port (meta server))))
      (log/info "Http-kit server is running at" uri)
      (assoc component :server server))")

  (stop [component] (log/info "Stopping HTTP Server")
    (server :timeout 100)
    (assoc component :server nil)))

(defn new-http-server [conf] (map->Httpserver {:conf conf}))
(defrecord Search [conf channels conn native-conn]
  component/Lifecycle
  (start [component]
    (log/info "Starting Search Component")
    (let [conn (esr/connect (:es-address conf))]
      (run-query-loop (:query channels) (:query-results channels) conf conn)
      (run-entity-details-loop (:entity-details channels) (:entity-details-results channels) conf conn)
      (assoc component :conn conn :native-conn native-conn :conn nil :native-conn nil)))
  (stop [component]
    (log/info "Stopping Search Component")
    (assoc component :conn nil :native-conn nil)))

(defn new-search [conf] (map->Search {:conf conf}))

(defrecord Search-Channels []
  component/Lifecycle
  (start [component] (log/info "Starting Search Channels Component")
    (assoc component
      :query (chan)
      :query-results (chan)
      :entity-details (chan)
      :entity-details-results (chan)
    ))
  (stop [component] (log/info "Stop Search Channels Component")
    (assoc component :query nil :query-results nil)))

(defn new-search-channels [] (map->Search-Channels {}}))
**core** handles communication and app state

**ui** OM components

**channels** just the definitions
(def app-state (atom {:query-string ""
   :aggs {:links {:buckets []}}
   :abstractions (list)
   :synonyms (list)
   :links (list)
   :ilinks (list)
   :text ""
   :linkage (list)
   :paths (list)}))

(def packer
  "Defines our packing (serialization) format for client<->server comms."
(sente-transit/get-flexi-packer :json))
(defn event-handler [{:keys [event]}]
  (match event
    [:chsk/state {:first-open? true}] (print "Socket established!"
    [:chsk/recv payload]
  (let [[msg-type msg] payload]
    (match [msg-type msg]
      [:cmd/linker-result hits] (put! c/linker-result-channel hits)
      [:cmd/entity-details-result hits] (put! c/entity-details-channel hits)
      [:cmd/graph-result hits] (put! c/graph-result-channel hits))
    :else (print "Unmatched event: %s" event)))
; Go baby
(go-loop []
  (let [linker-results (! c/linker-result-channel)]
    (swap! app-state assoc :linker-results linker-results)
    (recur)))

(go-loop []
  (let [graph-results (! c/graph-result-channel)]
    (swap! app-state update-in [:paths] conj [:path graph-results])
    (recur)))

(go-loop []
  (let [entity-details (! c/entity-details-channel)]
    (swap! app-state assoc :abstractions (:sig_categories document))
    (swap! app-state assoc :synonyms (:like_title document))
    (swap! app-state assoc :links (:links document))
    (swap! app-state assoc :ilinks (:ilinks document))
    (swap! app-state assoc :persons (:persons entity-details))
    (swap! app-state assoc :locations (:locations entity-details))
    (swap! app-state assoc :companies (:companies entity-details))
    (recur)))
A BIT CODE
& FRONT
LEARNINGS

* Structure clojure code via components

* Same language full stack

* Async communication from front to back

* use favorite datastores with clojure

* learn about wikipedia

& FIN
SHOULDERS

https://github.com/ elasticsearch/ elasticsearch-river-wikipedia

https://github.com/ OpenSextant/ SolrTextTagger

https://github.com/ stuartsierra/ component

https://github.com/ clojure/ core.async

https://github.com/ swannodette/ om

https://github.com/ matthiasn/ BirdWatch