Treball de Fi de Grau

Grau en Enginyeria en Tecnologies Industrials

Desenvolupament d'una aplicació web per a NLP

ANNEXOS

Autor: Max Montoliu Torruella
Director: Alexandre Perera Lluna
Convocatòria: Juliol 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

ANNEX A

```
def clean_frase(line, stop=False, inp= False):


# cle a n the terms in di c by :
# - lower a l l l e t t e r s
# - conve r t numbers in l e t t e r
# - s e p a r a t e words divided by a -
# - elimi n a t e punc tu a tion and o the r non p ri n t a bl e c h a r a c t e r s
# - i f Stop = True , elimi n a t e s stopwords
# - i f inp = t ru e i t al s o sep a r a ted coupled words (w1-w2) t h a t a re no t in
# the l i s t coupled terms and i t j o i n the one t h a t a re

  import re
  import string
  import nltk

  from unicodedata import normalize
  from num2words import num2words

  cleaned = []
  # prepare regez for char filtering
  re_print = re.compile('[^%s]' % re.escape(string.printable))

  remove = string.punctuation
  remove = remove.replace("-","") #don't remove hyphens
  table = str.maketrans(remove, ' '*len(remove))
  #normalize unicode characters (no vec que fa m'ho puc estalviar al haver aconseguit lines ja oi??)
  line = normalize('NFD', line).encode('ascii','ignore')
  line = line.decode('UTF-8')

  #convert to lowercase
  line = line.lower()


  #translate ordinal number (e.g: 1st ---> first)
  re_results = re.findall('(\d+(st|nd|rd|th))',line)
  if re_results:
    for enitre_result, suffix in re_results:
      num = int(enitre_result[:-2])
      tmp = num2words(num, ordinal=True)
      tmp += ' '
      line = line.replace(enitre_result, tmp)
  #convert numb-numb couples number to string (e.g.: 2-3 --> two three)
  re_results = re.findall("\d[\d]*\d[d]*", line)
  if re_results:
    for enitre_result in re_results:
      re_results2 = re.findall('\d[\d]*',enitre_result)
      string_tmp = ' '
      for num in re_results2:
        tmp = num2words(float(num), ordinal = True)
        tmp += ' '
        string_tmp += tmp
      line = re.sub(enitre_result, string_tmp, line)

  #convert all other numbers to string
  re_results = re.findall(" [-+]?[.]?[\d]+(?:,\d\d\d)*[\.]?\d(?:[eE][-+]?\d+)? ",line)
  if re_results:
    re_results = sorted(re_results, key= len)
    for  enitre_result in reversed(re_results):
      try:
        num = float(enitre_result)
        tmp = num2words(num)
```

```
def generar_vocab(corpus,freq_min = 1):
  '''''
  Input:
  - corpus:llista de llistes per cada terme
  -freq_min
  Tokenitzo les els termes del corpus. Em carrego els menys frequents
  '''
  print("Freqüencia mínima del diccionari:",freq_min)
  freq_dict =dict()
  for frase in corpus:
    for paraula in frase:
      if paraula in freq_dict:
        freq_dict[paraula]+=1
      else:
        freq_dict[paraula]=1

  tokens_freq = [tokens for tokens in freq_dict if freq_dict[tokens]>= freq_mi
n]
  idx = range(2,len(tokens_freq)+2)# el 0 serà per paraules desconegudes
  vocab = dict(zip(tokens_freq,idx))
  vocab
  vocab['pad'] = 0
  vocab['desc']= 1
  return freq_dict,vocab
```

```
def genera_tokens_processats(Terme, clean_frase = clean_frase):
  '''''
  A partir d'un terme de l'ontologia genera un text no processat amb la inform
ació necessària:
    * Nom
    * Definició
    * Sinònims
    * Pares
  Input: Term
  Output: llista de les paraules preprocessades
  '''

  it1 ="" if not Terme.synonyms else  map(lambda Synonym: Synonym.description
, Terme.synonyms)
  it2 ="" if not Terme.superclasses(1) else map(lambda Pare: Pare.name, Terme.
superclasses(1))
  frase=   " ".join(["" if not Terme.name else Terme.name ,"" if not Terme.def
inition else Terme.definition, " ".join(it1) ," ".join(it2)])
  frase = " ".join(["" if not Terme.name else Terme.name ,"" if not Terme.defi
nition else Terme.definition])
  frase_pre = clean_frase(frase)
  return frase_pre.split()
```

```python
def genera_train_i_vocab():
  from pronto import Ontology
  import random
  import numpy as np
  from tensorflow.keras.preprocessing.sequence import pad_sequences

  ls_ids =  ['HP:0000707','HP:0000924','HP:0040064']
  X_y = []
  ms = Ontology("hp.owl")
  for i in range(len(ls_ids)):
    n = ms[ls_ids[i]]
    X_y += [(i,genera_tokens_processats(node)) for node in n.subclasses()]

  y, corpus = list(zip(*X_y))## Aquí s'hauria d'aleatorietzar
  max_tokens = get_max(corpus)

  print('Corpus:',len(corpus))
  print('Frase més llarga (num):',max_tokens)

  freq,vocabulari = generar_vocab(corpus,freq_min = 1)
  desa_vocabulari(vocabulari,path='./')

  y_train = np.array(y)
  print('Mida y_train:',y_train.shape)
  corpus_tokenitzat =  list(map( lambda terme: [vocabulari[i] for i in  terme if i in vocabulari.keys() ],corpus ))
  X_train = pad_sequences(corpus_tokenitzat,maxlen=max_tokens,padding='post',truncating='post')
  print('Mida X_train:',X_train.shape,'-numtokens i max_tokens')

  return  X_train, y_train
```

```python
def model2(num_tokens=8501,max_tokens=119):
  from tensorflow.keras.layers import (GRU, Dense, Dropout, Embedding, Flatten,
                                       Input, Multiply, Permute, RepeatVector,
                                       Softmax,Embedding)
  from tensorflow.keras.models import Sequential

  from tensorflow.keras.utils import to_categorical
  from tensorflow.keras.optimizers import Adam
  '''''
  definem model
  '''
  model = Sequential()
  model.add(Embedding(input_dim= num_tokens,
                      output_dim =  EMBEDDING_SIZE,
                      input_length = max_tokens,
                      name='layer_embedding'))
  model.add(GRU(units=16, name = 'gru_1',return_sequences=True))
  model.add(GRU(units=8, name='gru_2',return_sequences=True))
  model.add(GRU(units=4,name='gru_3'))
  model.add(Dense(units = 3,activation='softmax',name="dense_1"))

  print(model.summary())
  return model
```

```python
def model1(num_tokens=8501,max_tokens=119):
  from tensorflow.keras.layers import (LSTM, Dense, Dropout, Embedding, Flatten, Softmax,Embedding)
  from tensorflow.keras.models import Sequential

  EMBEDDING_SIZE = 8


  #si s'executa més d'un cop petem els layers s'hauria de sistematitzar
  model = Sequential()
  model.add(Embedding(input_dim= num_tokens,
                      output_dim =  EMBEDDING_SIZE,
                      input_length = max_tokens,
                      name='layer_embedding'))#l'embeding hauria de sortir fora és veu clar amb lo de les GRU
  model.add(LSTM(units=16, name = 'lstm_1',return_sequences=True))
  model.add(LSTM(units=8, name = 'lstm_2',return_sequences=True))
  model.add(LSTM(units=4, name = 'lstm_3'))
  model.add(Dense(units = 3,activation='softmax',name="dense_1"))

  return model
```

```python
def desa_vocabulari(vocabulari,path='./'):
  import os
  import csv

  path_fitxer = os.path.join(path,'vocabulari.csv')
  f = open(path_fitxer, 'w')

  with f:
      #fnames = ['first_name', 'last_name'], fieldnames=fname
      writer = csv.writer(f)
      for i in vocabulari.keys():
        writer.writerow([i, vocabulari[i] ])
      print('Vocabulari desat a:',path_fitxer)
```

```python
def export_model(model,path='./model2'):
  import tensorflowjs as tfjs
  tfjs.converters.save_keras_model(
        model,
        path#'./model2'
        )
```

Resultats de l'entrenament:

Model amb LSTM

```
Epoch 1/15
213/213 [==============================] - 26s 123ms/step - loss: 1.0009 - accuracy: 0.5354 - val_loss: 1.8799 - val_accuracy: 0.0000e+00
Epoch 2/15
213/213 [==============================] - 26s 124ms/step - loss: 0.9866 - accuracy: 0.5354 - val_loss: 1.9221 - val_accuracy: 0.0000e+00
Epoch 3/15
213/213 [==============================] - 26s 122ms/step - loss: 0.9868 - accuracy: 0.5354 - val_loss: 1.8668 - val_accuracy: 0.0000e+00
Epoch 4/15
213/213 [==============================] - 27s 127ms/step - loss: 0.9869 - accuracy: 0.5354 - val_loss: 1.8599 - val_accuracy: 0.0000e+00
Epoch 5/15
213/213 [==============================] - 28s 132ms/step - loss: 0.9867 - accuracy: 0.5354 - val_loss: 1.8772 - val_accuracy: 0.0000e+00
Epoch 6/15
213/213 [==============================] - 27s 127ms/step - loss: 0.9864 - accuracy: 0.5354 - val_loss: 1.8201 - val_accuracy: 0.0000e+00
Epoch 7/15
213/213 [==============================] - 27s 128ms/step - loss: 0.9865 - accuracy: 0.5354 - val_loss: 1.9159 - val_accuracy: 0.0000e+00
Epoch 8/15
213/213 [==============================] - 28s 130ms/step - loss: 0.9866 - accuracy: 0.5354 - val_loss: 1.8776 - val_accuracy: 0.0000e+00
Epoch 9/15
213/213 [==============================] - 27s 128ms/step - loss: 0.9862 - accuracy: 0.5354 - val_loss: 1.8337 - val_accuracy: 0.0000e+00
Epoch 10/15
213/213 [==============================] - 28s 130ms/step - loss: 0.9866 - accuracy: 0.5354 - val_loss: 1.8510 - val_accuracy: 0.0000e+00
Epoch 11/15
213/213 [==============================] - 28s 130ms/step - loss: 0.9866 - accuracy: 0.5354 - val_loss: 1.8738 - val_accuracy: 0.0000e+00
Epoch 12/15
213/213 [==============================] - 28s 132ms/step - loss: 0.9863 - accuracy: 0.5354 - val_loss: 1.8509 - val_accuracy: 0.0000e+00
Epoch 13/15
213/213 [==============================] - 27s 127ms/step - loss: 0.9864 - accuracy: 0.5354 - val_loss: 1.8618 - val_accuracy: 0.0000e+00
Epoch 14/15
213/213 [==============================] - 27s 128ms/step - loss: 0.9862 - accuracy: 0.5354 - val_loss: 1.8700 - val_accuracy: 0.0000e+00
Epoch 15/15
213/213 [==============================] - 28s 131ms/step - loss: 0.9865 - accuracy: 0.5354 - val_loss: 1.8633 - val_accuracy: 0.0000e+00
```

Model amb GRU's

```
Epoch 1/15
213/213 [==============================] - 32s 152ms/step - loss: 1.0090 - accuracy: 0.5354 - val_loss: 1.7434 - val_accuracy: 0.0000e+00
Epoch 2/15
213/213 [==============================] - 32s 150ms/step - loss: 0.9875 - accuracy: 0.5354 - val_loss: 1.9277 - val_accuracy: 0.0000e+00
Epoch 3/15
213/213 [==============================] - 32s 151ms/step - loss: 0.9874 - accuracy: 0.5354 - val_loss: 1.8799 - val_accuracy: 0.0000e+00
Epoch 4/15
213/213 [==============================] - 32s 151ms/step - loss: 0.9864 - accuracy: 0.5354 - val_loss: 1.9175 - val_accuracy: 0.0000e+00
Epoch 5/15
213/213 [==============================] - 32s 150ms/step - loss: 0.9869 - accuracy: 0.5354 - val_loss: 1.8833 - val_accuracy: 0.0000e+00
Epoch 6/15
213/213 [==============================] - 32s 149ms/step - loss: 0.9867 - accuracy: 0.5354 - val_loss: 1.8592 - val_accuracy: 0.0000e+00
Epoch 7/15
213/213 [==============================] - 32s 150ms/step - loss: 0.9866 - accuracy: 0.5354 - val_loss: 1.8271 - val_accuracy: 0.0000e+00
Epoch 8/15
213/213 [==============================] - 31s 148ms/step - loss: 0.9871 - accuracy: 0.5354 - val_loss: 1.8683 - val_accuracy: 0.0000e+00
Epoch 9/15
213/213 [==============================] - 32s 151ms/step - loss: 0.9867 - accuracy: 0.5354 - val_loss: 1.8992 - val_accuracy: 0.0000e+00
Epoch 10/15
213/213 [==============================] - 33s 153ms/step - loss: 0.9870 - accuracy: 0.5356 - val_loss: 1.8675 - val_accuracy: 0.0000e+00
Epoch 11/15
213/213 [==============================] - 32s 152ms/step - loss: 0.9864 - accuracy: 0.5356 - val_loss: 1.8513 - val_accuracy: 0.0000e+00
Epoch 12/15
213/213 [==============================] - 32s 151ms/step - loss: 0.9860 - accuracy: 0.5356 - val_loss: 1.8368 - val_accuracy: 0.0000e+00
Epoch 13/15
213/213 [==============================] - 32s 150ms/step - loss: 0.9473 - accuracy: 0.5768 - val_loss: 2.0229 - val_accuracy: 0.0000e+00
Epoch 14/15
213/213 [==============================] - 32s 149ms/step - loss: 0.5461 - accuracy: 0.8118 - val_loss: 1.7481 - val_accuracy: 0.0000e+00
Epoch 15/15
213/213 [==============================] - 32s 151ms/step - loss: 0.4523 - accuracy: 0.8304 - val_loss: 1.7023 - val_accuracy: 0.0000e+00
```

Carrega.js

```
    // el carrego així per poder jugar-hi des del terminal d'inspecció
    async function carregaModel() {
    const MODEL_URL = 'http://127.0.0.1:8887/modeljs/model.json';
    model = await tf.loadLayersModel(MODEL_URL);
    console.log(model.summary());
    return model
};
    async function carregaVocabulari()
    {
      await $.ajax({
      url: 'http://127.0.0.1:8887/vocabulari.csv',
      dataType: 'text',
      crossDomain : true}).done(success);
    };
function success(data) {
    var wd_idx = new Object();
    lst = data.split(/\r?\n|\r/);
    for (var i = 0; i < lst.length; i++) {
        key = (lst[i]).split(',')[0]
        value = (lst[i]).split(',')[1]

        if (key == "")
            continue
        wd_idx[key] = parseInt(value)
    }
    word_index = wd_idx
}

function process(txt){
    txt= txt.replace(/[^a-zA-Z0-9\s]/, '')// s'ha de valorar
    txt= txt.trim().split(/\s+/)
    for (let i = 0;i < txt.length;i++)
        txt[i] = txt[i].toLowerCase();
    return txt// ja tenim l'array
}
const MAX_TOKENS = 119;
function tokensaSeq(txt){
    txt= process(txt);
    let sequence= [];
    txt.forEach(function(word){
        let id = word_index[word];
        if (id == undefined){
            console.log(id);//per si incloem unknown més endavant
            sequence.push(0)
        }else{
            sequence.push(id);
        }});
        //afegim el padding
    if (sequence.length < MAX_TOKENS){
        let pad_array = Array(MAX_TOKENS - sequence.length)
        pad_array.fill(0);
        sequence = [...sequence,...pad_array];

    }
    return sequence
}

async function  pre_predict(){
    carregaVocabulari()
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@latest"></script>
    <script src="https://cdn.jsdelivr.net/gh/nicolaspanel/numjs@0.15.1/dist/numjs.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <script src="./carrega.js"></script>
    <meta charset="UTF-8">
    <title>Model LSTM</title>
</head>
<body>

    <form id="form" onsubmit="return false;">
        <input style="position:absolute; top:20%; left:5%; width:70%;  line-height: 4em;" type="text" id="userInput" />
        <input style="position:absolute; top:35%; left:5%; width:70%;" type="submit" onclick="predict();" />
    </form>
    <div id="output1"></div>
    <div id="output2"></div>
</body>
</html>
<script >

//prepararem model i diccionari
let word_index;
let model;
var t0 = performance.now()
pre_predict()// no és un problema de moment
var t1 = performance.now()
console.log("Tarda: " + (t1 - t0) + " millisegons.")
alert("Ja s'ha carregat allò necessàri per ajudar-lo. Expliqui'ns que passa")


let pred;

async function predict() {
    let tp1= performance.now()
    txt = document.getElementById("userInput").value;
    seq = tokensaSeq(txt)
    console.log("seqüencia input:")
    console.log(seq);
    let iTensor = tf.tensor1d(seq, dtype = 'int32').expandDims(0);
    pred = model.predict(iTensor)
    document.getElementById("output1").innerText = hposMespropables(pred);
    let tp2 = performance.now()
    alert("Input tractat:" + txt+ " \nPredicció efectuada en: "+(tp2-tp1)+" mil·lisegons")
}
let hpoArray = ['HP:0000707','HP:0000924','HP:0040064'];

function hpoMesProbable(predictions){
     const predResult = predictions.dataSync();
     const idx = predResult.indexOf(Math.max(...predResult));//pot fer-se amb tf segur
     return hpoArray[idx]
}
```

```
54. async function hposMespropables(predictions){
55.     const soglia = 0;
56.     const probabilitats = predictions.dataSync();
57.     const rati = tf.tensor1d([soglia])
58.     const mask = tf.squeeze(predictions.greater(rati));
59.     mask.print()
60.
61.     idxS = await  tf.booleanMaskAsync(tf.range(0,hpoArray.length,1,dtype='int3
    2'),mask);
62.     const outp = document.getElementById('output1');//apunto al div o2
63.     outp.innerHTML = "<br>Amb prob superior a :"+ soglia;
64.     for(const i of idxS.dataSync()){
65.         outp.innerHTML +="<br>"+hpoArray[i] +" : "+ probabilitats[i];
66.     }
67.
68.
69. }
70. </script>
```

Annex B

```
def genera_jsons_vega(ls_id_nom)
  '''''
  Generar els fitxers json necessàris per a la visualització del graf radial de vega
  Input: LLista de tuples amb id i noms
  Output: Fitxers jsons amb nom de l'HPO que representent
  '''
  import json
  ls_css_files = []
  for id,nom in ls_id_nom:
    nom = 'HP'+id[3:]+'.json'
    ls_css_files.append(nom)
    data_id = Graf_radial(ms,id)
    with open(nom, 'w') as fp:
      print("escrivint fitxer {}".format(nom))
      json.dump(data_id, fp,indent=4)
```

```
def Graf_radial(ms,id):
  import json
  import urllib.request
  from altair import VegaLite

  instruments = set(ms[id].subclasses())
  data = []
  for term in instruments:
    value = {"id": int(term.id[3:]), "name":term.id,"desc":term.name}
    parents = instruments.intersection(term.relationships.get(ms['is_a'],set()))
    if parents:
      value['parent']=int(parents.pop().id[3:])
    data.append(value)

  # Let's use the Vega radial tree example as a basis of the visualization
  view = json.load(urllib.request.urlopen("https://vega.github.io/vega/examples/radial-tree-layout.vg.json"))
  # First replace the default data with our own
  print(view['data'][0].pop('url'))
  view['data'][0]['values'] = data
  view['marks'][1]['encode']['enter']['tooltip'] = {"signal": "datum.desc"}

  view['signals'][4]['value'] = 'cluster'
  # Render the clustered tree
  return view
```

```
1.  def terme2json(id):
2.    """
3.    Funció per a generar l'estructura json necessària per al dendograma
4.    """
5.    from pronto import Ontology
6.    ms = Ontology("hp.owl")#carrego l'ontologia això només cal fer-ho un cop
7.    instruments = set(ms[id].subclasses())
8.    data=[]
9.    for term in instruments:
10.     value = [ int(term.id[3:]),term.name]
11.     parents = instruments.intersection(term.relationships.get(ms['is_a'],set()
    ))
12.     if parents:#no és el node arrel
13.       value.append(int(parents.pop().id[3:]))
14.     else:
15.       value.append('')
16.     data.append(value)
17.   data
18.   d = {"data":data}
19.   return data
```

ANNEX C

```
{
  "$id": "https://example.com/input.schema.json",
  "$schema": "https://json-schema.org/draft/2019-09/schema",
  "title": "Predictor Output",
  "type": "object",
  "properties": {
    "data": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "id": {
              "type": "integer",
              "description": "Numeric index of the prediction."
            },
            "data": {
              "type": "array",
              "description": "The data associated with the prediction subgraph.",
              "items": [
                {
                  "type": "object",
                  "properties": {
                    "name": {
                      "type": "string",
                      "description": "Name of the parent (strictly speaking children) node of the predicted node."
                    },
                    "children": {
                      "type": "array",
                      "description": "Name of the children (strictly speaking parent) nodes of the top-level node.",
                      "items": [
                        {
                          "type": "object",
                          "properties": {
                            "is_predicted": {
                              "type": "boolean",
                              "description": "if True this is the predicted node."
                            },
                            "name": {
                              "type": "string",
                              "description": "Name of the node(s) at the second level (including predicted)."
                            },
                            "children": {
                              "type": "array",
                              "description": "Name of the children (strictly speaking parent) nodes of the second-level nodes (including predicted)."
                            }
                          },
                          "required": [ "is_predicted", "name" ]
                        }
                      ],
                      "minItems": 1,
                      "uniqueItems": true
                    }
                  },
                  "required": [ "name", "children" ]
                }
              ],
              "minItems": 1,
              "uniqueItems": true
            }
```

```
{
  "$id": "https://example.com/input.schema.json",
  "$schema": "https://json-schema.org/draft/2019-09/schema",
  "title": "Predictor Input",
  "type": "object",
  "properties": {
    "content": {
      "type": "string",
        "description": "The content introduced in the text box that needs to be translated.",
        "minLength": 2
      },
      "limit": {
        "type": "integer",
        "description": "Length of the returned list.",
        "minimum": 1
      }
    },
    "required": [ "content", "limit" ]
}
```