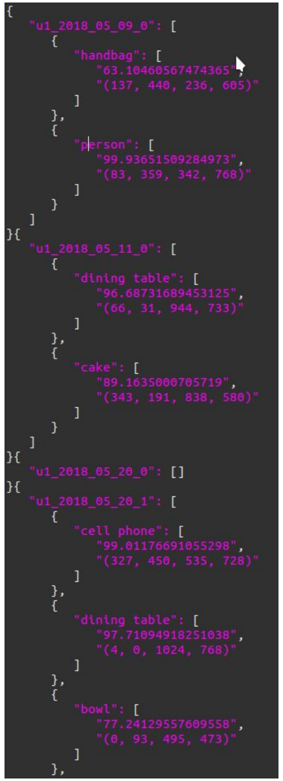
Reunião 12/02/20

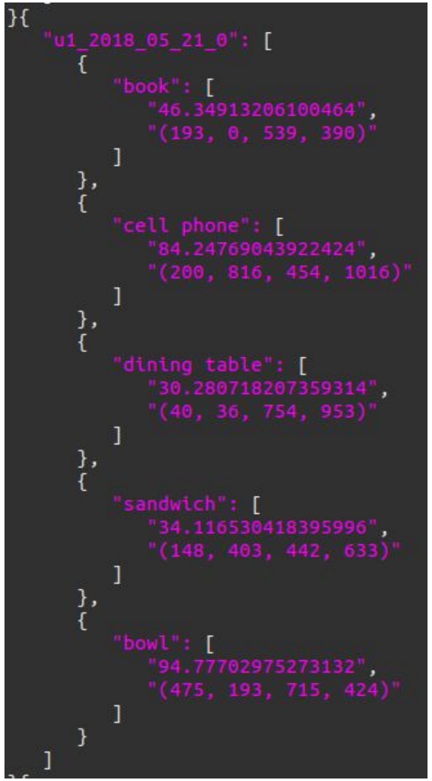
Objetivos propostos (reunião 21/01/20):

1. Ver site imageclef.
2. Dar ID às imagens nos ficheiros json.
3. Procurar formas de combinar palavras (artigos, algoritmos).
4. Ver tema : Natural Language Processing e Word Embedding.
5. Procurar por SpaCy e Gensim.

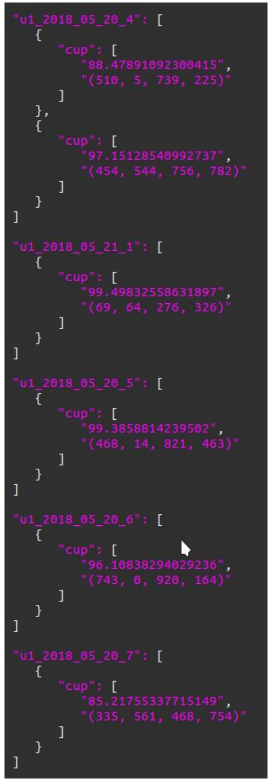
ID dado às imagens no ficheiro json.



```
{
   "u1_2018_05_09_0": [
      {
         "handbag": [
            "63.10460567474365",
            "(137, 440, 236, 605)"
         ]
      },
      {
         "person": [
            "99.93651509284973",
            "(83, 359, 342, 768)"
         ]
      }
   ]
}{
   "u1_2018_05_11_0": [
      {
         "dining table": [
            "96.68731689453125",
            "(66, 31, 944, 733)"
         ]
      },
      {
         "cake": [
            "89.1635000705719",
            "(343, 191, 838, 580)"
         ]
      }
   ]
}{
   "u1_2018_05_20_0": []
}{
   "u1_2018_05_20_1": [
      {
         "cell phone": [
            "99.01176691055298",
            "(327, 450, 535, 728)"
         ]
      },
      {
         "dining table": [
            "97.71094918251038",
            "(4, 0, 1024, 768)"
         ]
      },
      {
         "bowl": [
            "77.24129557609558",
            "(0, 93, 495, 473)"
         ]
      },
```



```
}{
   "u1_2018_05_21_0": [
      {
         "book": [
            "46.34913206100464",
            "(193, 0, 539, 390)"
         ]
      },
      {
         "cell phone": [
            "84.24769043922424",
            "(200, 816, 454, 1016)"
         ]
      },
      {
         "dining table": [
            "30.280718207359314",
            "(40, 36, 754, 953)"
         ]
      },
      {
         "sandwich": [
            "34.116530418395996",
            "(148, 403, 442, 633)"
         ]
      },
      {
         "bowl": [
            "94.77702975273132",
            "(475, 193, 715, 424)"
         ]
      }
   ]
```



```
"u1_2018_05_20_4": [
   {
      "cup": [
         "88.47891092300415",
         "(510, 5, 739, 225)"
      ]
   },
   {
      "cup": [
         "97.15128540992737",
         "(454, 544, 756, 782)"
      ]
   }
]

"u1_2018_05_21_1": [
   {
      "cup": [
         "99.49832558631897",
         "(69, 64, 276, 326)"
      ]
   }
]

"u1_2018_05_20_5": [
   {
      "cup": [
         "99.3858814239502",
         "(468, 14, 821, 463)"
      ]
   }
]

"u1_2018_05_20_6": [
   {
      "cup": [
         "96.10838294029236",
         "(743, 0, 920, 164)"
      ]
   }
]

"u1_2018_05_20_7": [
   {
      "cup": [
         "85.21755337715149",
         "(335, 561, 468, 754)"
      ]
   }
]
```

Natural Language Processing Libraries

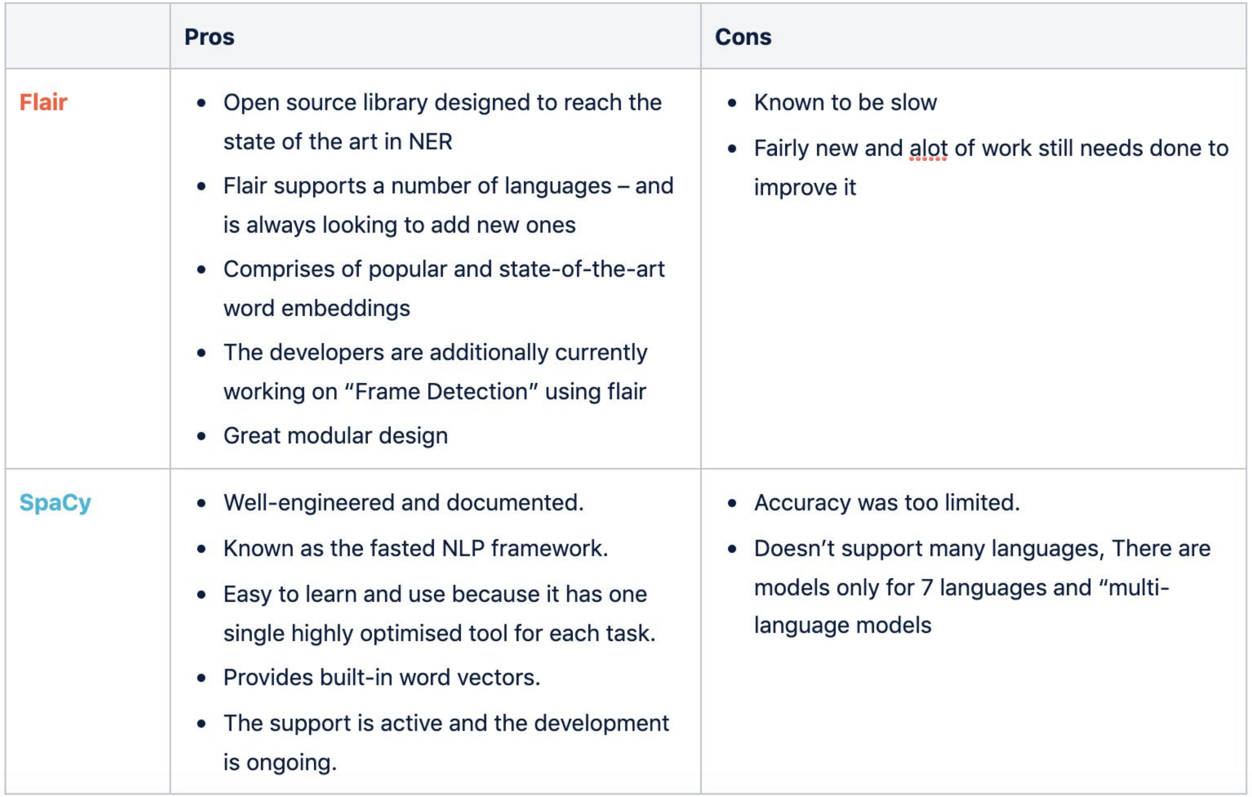
- **SpaCy** → Provavelmente a melhor para usar.
- Natural Language Toolkit (NLTK)
- Pattern
- Gensim
- Stanford Core NLP
- Scikit – learn
- **Flair** → Atualmente considerada S.O.T.A
- PolyGlot

Comparison of Python NLP libraries Pros and Cons

	⊕ PROS	⊖ CONS
Natural Language ToolKit	+ The most well-known and full NLP library + Many third-party extensions + Plenty of approaches to each NLP task + Fast sentence tokenization + Supports the largest number of languages compared to other libraries	- Complicated to learn and use - Quite slow - In sentence tokenization, NLTK only splits text by sentences, without analyzing the semantic structure - Processes strings which is not very typical for object-oriented language Python - Doesn't provide neural network models - No integrated word vectors
spaCy	+ The fastest NLP framework + Easy to learn and use because it has one single highly optimized tool for each task + Processes objects; more object-oriented, comparing to other libs + Uses neural networks for training some models + Provides built-in word vectors + Active support and development	- Lacks flexibility, comparing to NLTK - Sentence tokenization is slower than in NLTK - Doesn't support many languages. There are models only for 7 languages and "multi-language" models
scikit learn NLP toolkit	+ Has functions which help to use the bag-of-words method of creating features for the text classification problems + Provides a wide variety of algorithms to build machine learning models + Has good documentation and intuitive classes' methods	- For more sophisticated preprocessing things (for example, pos-tagging), you should use some other NLP library and only after it you can use models from scikit-learn - Doesn't use neural networks for text preprocessing
gensim	+ Works with large datasets and processes data streams + Provides tf-idf vectorization, word2vec, document2vec, latent semantic analysis, latent Dirichlet allocation + Supports deep learning	- Designed primarily for unsupervised text modeling - Doesn't have enough tools to provide full NLP pipeline, so should be used with some other library (Spacy or NLTK)
Pattern	+ Allows part-of-speech tagging, n-gram search, sentiment analysis, WordNet, vector space model, clustering and SVM + There are web crawler, DOM parser, some APIs (like Twitter, Facebook etc.)	- Is a web miner; can be not enough optimized for some specific NLP tasks
Polyglot	+ Supports a large number of languages (16-196 languages for different tasks)	- Not as popular as, for example, NLTK or Spacy; can be slow issues solutions or weak community support

Created by ActiveWizards

SpaCy vs Flair



	Pros	Cons
Flair	• Open source library designed to reach the state of the art in NER • Flair supports a number of languages – and is always looking to add new ones • Comprises of popular and state-of-the-art word embeddings • The developers are additionally currently working on "Frame Detection" using flair • Great modular design	• Known to be slow • Fairly new and alot of work still needs done to improve it
SpaCy	• Well-engineered and documented. • Known as the fasted NLP framework. • Easy to learn and use because it has one single highly optimised tool for each task. • Provides built-in word vectors. • The support is active and the development is ongoing.	• Accuracy was too limited. • Doesn't support many languages, There are models only for 7 languages and "multi-language models

Word Embeddings.

Static Word Embedding algorithms

Estes algoritmos não conseguem fazer a distinção da utilização da mesma palavra em contextos diferentes (cada palavra tem apenas um vetor, independentemente do contexto).

Por exemplo:

- “This **apple** is delicious”
- “The **Apple** company sells the iPhone”.

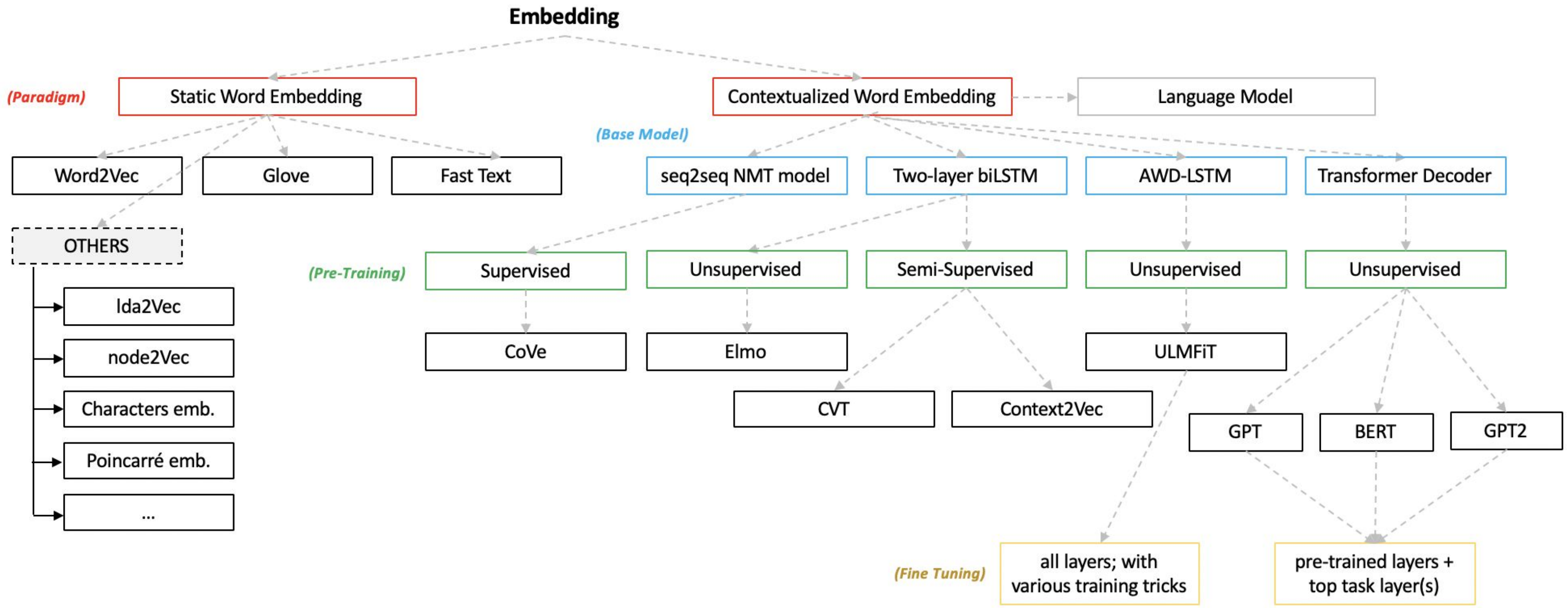
- ➔ Word2Vec (Skip-Gram and CBOW).
- ➔ Glove.
- ➔ FastText.

Word Embeddings

Contextualized Word Embeddings Algorithms

- ➔ COVE
- ➔ CUT
- ➔ ELMO
- ➔ ULMFit
- ➔ BERT
- ➔ GPT and GPT2
- ➔ **XLNet** ⟶ Current S.O.T.A
- ➔ Flair Embeddings
- ➔ ENRIE

©AdrienSIEG


Embedding
(Paradigm)
Static Word Embedding
Contextualized Word Embedding
Language Model
Word2Vec
Glove
Fast Text
OTHERS
lda2Vec
node2Vec
Characters emb.
Poincarré emb.
...
(Base Model)
seq2seq NMT model
Two-layer biLSTM
AWD-LSTM
Transformer Decoder
(Pre-Training)
Supervised
Unsupervised
Semi-Supervised
Unsupervised
Unsupervised
CoVe
Elmo
CVT
Context2Vec
ULMFiT
GPT
BERT
GPT2
(Fine Tuning)
all layers; with various training tricks
pre-trained layers + top task layer(s)

(Extra) Como alguns papers fizeram no imageclef 2019

A Multimedia Modular Approach to Lifelog Moment Retrieval (nome PDF)

- ➔ Utilizaram a ferramenta **python - rake** (python module for Rapid Automatic Keyword Extraction Algorithm.
  - ◆ (Também existe a ferramenta python-rake-nltk que faz rapid automatic keyword using NLTK (Natural Language Toolkit).
- ➔ Utilizaram wordNet para gerar sinónimos pré-definidos.
- ➔ Utilizaram word2Vec para, também, gerar sinónimos pré-definidos.

Keywords :

➔ For each topic they extracted the associated keywords with Python-Rake package. Python-Rake cuts the sentence into words, filters with a list of stop-words and returns the remain keywords.

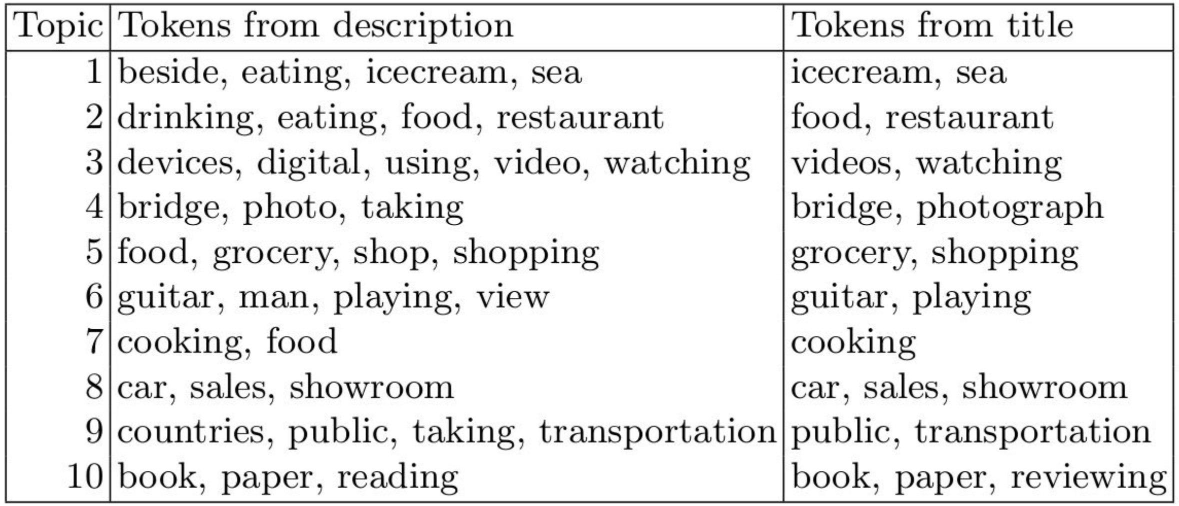
Synonyms:

➔ This method helps to diversify the topic representation by adding synonyms to the basic extracted keywords.

WordNet is used to generate predefined synonyms based on the wordNet hierarchy. The similarity was computed by **GoogleNews-Vector-Negative-300** which is a word2Vec model.
Another method to compute similarity uses the vector representation of each words in a word2Vec model. This similarity is useful to filter irrelevant synonyms. Th final method consists of selecting synonyms from wordNet with a threshold of 0.5 on the word2Vec model similarity.

Automated Lifelog Moment Retrieval based on Image Segmentation and Similarity Scores (nome PDF)

1. **Query Source**: Query description and Query title.
2. **Removed punctuation from the query, lowercased it and tokenized it.**

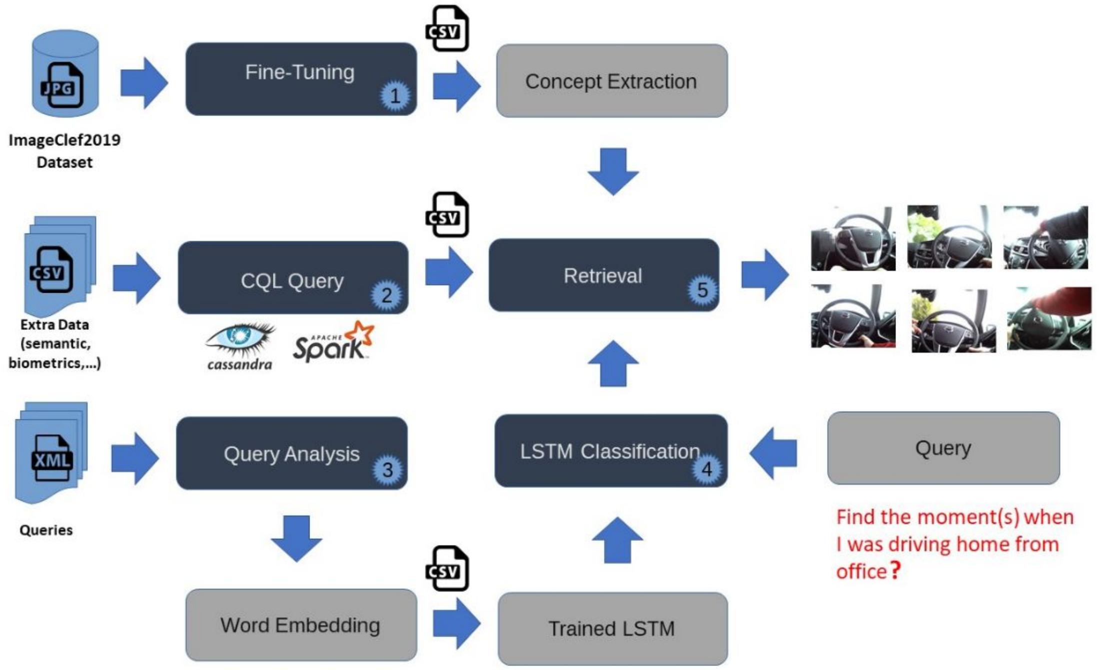


Topic	Tokens from description	Tokens from title
1	beside, eating, icecream, sea	icecream, sea
2	drinking, eating, food, restaurant	food, restaurant
3	devices, digital, using, video, watching	videos, watching
4	bridge, photo, taking	bridge, photograph
5	food, grocery, shop, shopping	grocery, shopping
6	guitar, man, playing, view	guitar, playing
7	cooking, food	cooking
8	car, sales, showroom	car, sales, showroom
9	countries, public, taking, transportation	public, transportation
10	book, paper, reading	book, paper, reviewing

3. **Token Vector**:
    - For each query taken, a vector with the same size as the image vector is created. The entries of the vector were the cosine similarities between the token and each of the labels retrieved from the word vector. These similarities were inside the interval [-1, 1], whereby 1 was the highest similarity.

4. **Word vectors** :
    - pre trained with **GloVe** Vectors from common crawl.

Big Data For Lifelog Moments Retrieval Improvement (nome PDF)



1. **Query analysis:** To extract relevant concepts from the given query, they built labeled textual descriptions of query moments.
2. **Test set topics:**
   a. NTCR-12
   b. NTCR-13
   c. Imageclef LRT 2017
   d. imageclef LRT 2016
3. Obtained a csv file which contains for each topic title, the topic description and the relevant concepts associated with the topic.
4. Convert the concepts to numeric vectors by training word embedding.