>>> IHTL final project
>>> Text similarity

Name: Gusssseppe Bravo and Arthur Müller
Date: December 12, 2019

>>> Approach Architecture

1. Real-valued distance metrics with different approaches:
   * Lexical-based
   * Syntactical-based
   * Sentence-Embeddings-based

	ps	lch	wup	lin	dp	infer	over	uni
0	0.576236	0.478867	0.603125	0.441224	0.944957	0.958609	0.642857	0.308706
1	0.433333	0.366896	0.444444	0.280471	0.916931	0.946287	0.631579	0.121527
2	0.423077	0.371224	0.485209	0.043715	0.939007	0.938343	0.500000	0.044599
3	0.622222	0.355331	0.638889	0.333333	0.950514	0.964189	0.733333	0.394504
4	0.621759	0.593266	0.727564	0.465101	0.879717	0.913712	0.260870	0.143134

2. And then apply Tunning and Stacking to Gradient Boosting models:
   * XGBRegressor
   * LightGBM

\>>> Lexical Features

Explore step-by-step:

* Remove stopwords and punctuation
* Lemmatize
* PoS tags
* WordNet synsets (best per lemma)

And then greedy lemma alignment with different distances:

ps Path similarity

lch Leacock Chodorow (normalized)

wup Wu Palmer

lin Lin

>>> Syntactical Features

Use CoreNLP:

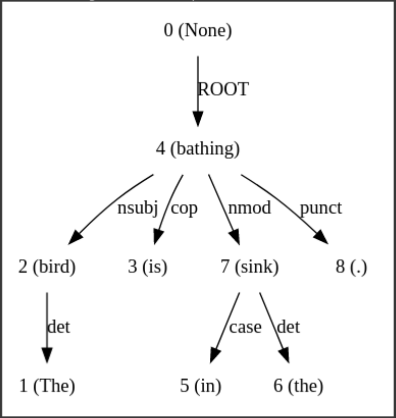
* Build dependency trees
* Filter unique governing and dependent lemmas
* Filter verbs and nouns with main content

dp Apply Infersent distance metrics (next slide)

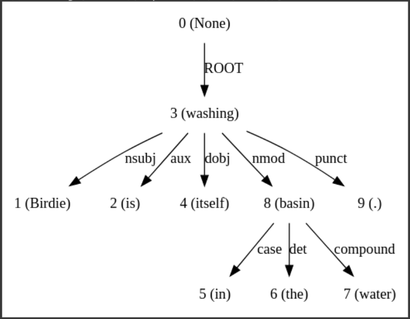
Assumption:

Dependency trees will provide more robust way of recognizing verbs and nouns because of relationship between words.

```
[(('bathing', 'NN'), 'nsubj', ('bird', 'NN')),
 (('bird', 'NN'), 'det', ('The', 'DT')),
 (('bathing', 'NN'), 'cop', ('is', 'VBZ')),
 (('bathing', 'NN'), 'nmod', ('sink', 'NN')),
 (('sink', 'NN'), 'case', ('in', 'IN')),
 (('sink', 'NN'), 'det', ('the', 'DT')),
 (('bathing', 'NN'), 'punct', ('.', '.'))]
```



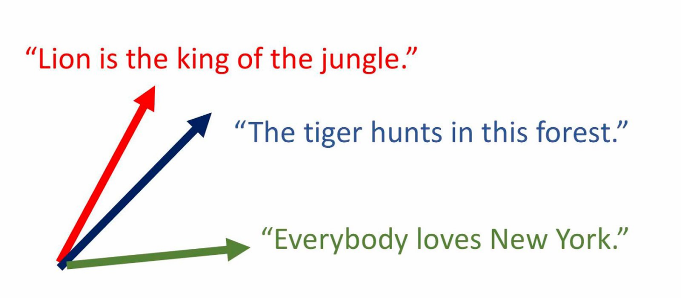

```
[(('washing', 'VBG'), 'nsubj', ('Birdie', 'NNP')),
 (('washing', 'VBG'), 'aux', ('is', 'VBZ')),
 (('washing', 'VBG'), 'dobj', ('itself', 'PRP')),
 (('washing', 'VBG'), 'nmod', ('basin', 'NN')),
 (('basin', 'NN'), 'case', ('in', 'IN')),
 (('basin', 'NN'), 'det', ('the', 'DT')),
 (('basin', 'NN'), 'compound', ('water', 'NN')),
 (('washing', 'VBG'), 'punct', ('.', '.'))]
```




```
[~]$ _
```

>>> Sentence Embeddings

* *Problematic: encode sentences with different lengths*
* If it had same length we can chain word2vec embeddings and that's it.
* If not, we can use CBOW or LSTM to deal with fixed representation.
* The basic idea is to remember each word in a sentence and train an encoder-decoder, so that, we can use the encoder to generate embeddings from its latent space.



>>> New Features

Word Embeddings models:

- **infer** Infersent
- **uni** Universal Sentence Encoder

Overlapping:

- **over** Simple word overlap as feature

Notice:

Preprocessing for word embeddings is similar to lexical features, but without WordNet senses.

>>> Evaluation

Base models:

XGBoost Score: 0.751

LightGBM Score: 0.749

Tuning (Hyperopt, Bayesian searcher):

XGBoost Score: 0.750

LightGBM Score: 0.759

Stacking (best two models):

Stacked Score: 0.771

\>>> Metrics & Features' Influence

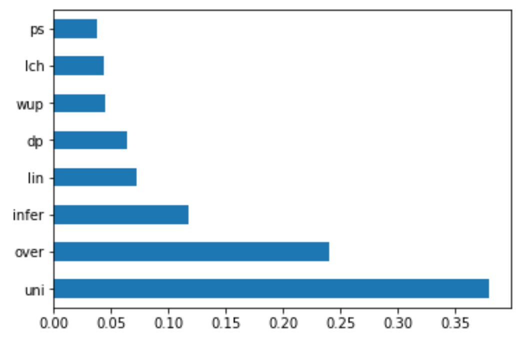
0.570 lexical alone

0.582 syntactical alone

0.662 lexical + syntetical

0.724 word embeddings + overlapping

0.771 all together



>>> Comparing to SemEval 2012

Best systems back in 2012:

0.824 UKP

0.814 TakeLab

Some reused approaches from TakeLab example:

* Greedy Lemma Alignment Overlap
* Dependency Trees, but other similarity metric
* Vector space sentence similarity (-> Word Embeddings)

However:

* TakeLab used far more features, e.g. exploiting of NERC
* UKP used e.g. Text Expansion Mechanisms to enrich sentences and alleviate lexical gaps

>>> Failed Approaches

* Blending
* Features selection
* AutoML: auto-sklearn, TPOT (GA)
* AutoHyperparameter: Hyperopt-sklearn

>>> Code

Notebook

* https://colab.research.google.com/drive/1Qb1XkJTMlXuH8P5AeLbxV9VvQ

Repository

* https://github.com/gusseppe/master_artificial_intelligence/tree/master/Introduction_to_Human_Language_Technology/deliverables/project

Thanks!