Relation extraction

Bill MacCartney
CS224u
Stanford University

Simple baselines

Overview

- ~~The task of relation extraction~~
- ~~Data resources~~
- ~~Problem formulation~~
- ~~Evaluation~~
- Simple baselines
- Directions to explore

Simple baselines

- Random guessing
- Common fixed phrases
- A simple classifier

Simple baselines

Random guessing

```
def random_classifier (xs):
    return [random.random()  < 0.5 for x in xs]

rel_ext.evaluate(splits, random_classifier, test_split  ='dev')
```

relation	precision	recall	f-score	support	size
adjoins	0.062	0.543	0.075	407	7057
author	0.095	0.519	0.113	657	7307
capital	0.019	0.508	0.023	126	6776
contains	0.402	0.501	0.419	4487	11137
film_performance	0.127	0.494	0.149	984	7634
founders	0.064	0.484	0.078	469	7119
genre	0.031	0.507	0.038	205	6855
has_sibling	0.085	0.494	0.102	625	7275
has_spouse	0.098	0.481	0.116	754	7404
is_a	0.085	0.503	0.102	618	7268
nationality	0.062	0.567	0.076	386	7036
parents	0.055	0.513	0.068	390	7040
place_of_birth	0.045	0.550	0.055	282	6932
place_of_death	0.030	0.502	0.037	209	6859
profession	0.044	0.500	0.054	308	6958
worked_at	0.041	0.472	0.050	303	6953
macro-average	0.084	0.509	**0.097**	11210	117610

It's good practice to start by evaluating a weak baseline like random guessing.

Recall is generally around 0.50.

Precision is generally poor.

F-score is generally poor.

(But look at `contains`!)

The number to beat: 0.097.

Common fixed phrases

Let's write code to find the most common middles for each relation.

```
def find_common_middles (split, top_k =3, show_output =False):
    corpus = split.corpus
    kb = split.kb
    mids_by_rel = {
        'fwd': defaultdict( lambda: defaultdict( int)),
        'rev': defaultdict( lambda: defaultdict( int))}
    for rel in kb.all_relations:
        for kbt in kb.get_triples_for_relation(rel):
            for ex in corpus.get_examples_for_entities(kbt.sbj, kbt.obj):
                mids_by_rel[ 'fwd'][rel][ex.middle] += 1
            for ex in corpus.get_examples_for_entities(kbt.obj, kbt.sbj):
                mids_by_rel[ 'rev'][rel][ex.middle] += 1
    def most_frequent (mid_counter):
        return sorted ([(cnt, mid) for mid, cnt in mid_counter.items()], reverse =True)[:top_k]
    for rel in kb.all_relations:
        for dir in ['fwd', 'rev']:
            top = most_frequent(mids_by_rel[dir][rel])
            if show_output:
                for cnt, mid in top:
                    print('{:20s} {:5s} {:10d} {:s}' .format(rel, dir, cnt, mid))
            mids_by_rel[dir][rel] = set([mid for cnt, mid in top])
    return mids_by_rel
```

Simple baselines

Common fixed phrases

```
_ = find_common_middles(splits['train'], show_output=True)
```

```
...
film_performance     fwd          283 in
film_performance     fwd          151 's
film_performance     fwd           96 film
film_performance     rev          183 with
film_performance     rev          128 , starring
film_performance     rev           97 opposite
...
has_sibling          fwd         1115 and
has_sibling          fwd          545 ,
has_sibling          fwd          125 , and
has_sibling          rev          676 and
has_sibling          rev          371 ,
has_sibling          rev           68 , and
...
parents              fwd           64 , son of
parents              fwd           45 and
parents              fwd           42 ,
parents              rev          187 and
parents              rev          151 ,
parents              rev           42 and his son
...
```

Simple baselines

Common fixed phrases

```
rel_ext.evaluate(splits, train_top_k_middles_classifier())
```

relation	precision	recall	f-score	support	size
adjoins	**0.272**	0.285	**0.274**	407	7057
author	**0.325**	0.078	**0.198**	657	7307
capital	0.089	0.159	0.097	126	6776
contains	0.582	0.064	0.222	4487	11137
film_performance	0.455	0.005	0.024	984	7634
founders	0.146	0.038	0.094	469	7119
genre	0.000	0.000	0.000	205	6855
has_sibling	**0.261**	0.176	**0.238**	625	7275
has_spouse	**0.349**	0.211	**0.309**	754	7404
is_a	0.068	0.024	0.050	618	7268
nationality	0.103	0.036	0.075	386	7036
parents	0.081	0.067	0.077	390	7040
place_of_birth	0.016	0.007	0.013	282	6932
place_of_death	0.024	0.014	0.021	209	6859
profession	0.039	0.039	0.039	308	6958
worked_at	0.050	0.020	0.038	303	6953
macro-average	0.179	0.076	**0.111**	11210	117610

Recall is much worse across the board.

But precision and F-score have improved for many relations, especially `adjoins`, `author`, `has_sibling`, and `has_spouse`.

The new number to beat: 0.111.

Simple baselines

A simple classifier: bag-of-words features

```
def simple_bag_of_words_featurizer(kbt, corpus, feature_counter):
    for ex in corpus.get_examples_for_entities(kbt.sbj, kbt.obj):
        for word in ex.middle.split(' '):
            feature_counter[word] += 1
    for ex in corpus.get_examples_for_entities(kbt.obj, kbt.sbj):
        for word in ex.middle.split(' '):
            feature_counter[word] += 1
    return feature_counter
```

A simple classifier: bag-of-words features

```
kbt = kb.kb_triples[0]

kbt
```

```
KBTriple(rel='contains', sbj='Brickfields', obj='Kuala_Lumpur_Sentral_railway_station')
```

```
corpus.get_examples_for_entities(kbt.sbj, kbt.obj)[0].middle
```

```
'it was just a quick 10-minute walk to'
```

```
simple_bag_of_words_featurizer(kb.kb_triples[0], corpus, Counter())
```

```
Counter({'it': 1,
         'was': 1,
         'just': 1,
         'a': 1,
         'quick': 1,
         '10-minute': 1,
         'walk': 1,
         'to': 2,
         'the': 1})
```

Simple baselines

A simple classifier: training a model

```
train_result = rel_ext.train_models(
    splits,
    featurizers=[simple_bag_of_words_featurizer],
    split_name='train',
    model_factory=(lambda: LogisticRegression(fit_intercept =True, solver='liblinear')))
```

A simple classifier: making predictions

```
predictions, true_labels  = rel_ext.predict(
    splits, train_result, split_name ='dev')
```

A simple classifier: evaluating predictions

```
rel_ext.evaluate_predictions(predictions, true_labels)
```

relation	precision	recall	f-score	support	size
adjoins	0.832	0.378	0.671	407	7057
author	0.779	0.525	0.710	657	7307
capital	0.638	0.294	0.517	126	6776
contains	0.783	0.608	0.740	4487	11137
film_performance	0.796	0.591	0.745	984	7634
founders	0.783	0.384	0.648	469	7119
genre	0.654	0.166	0.412	205	6855
has_sibling	0.865	0.246	0.576	625	7275
has_spouse	0.878	0.342	0.668	754	7404
is_a	0.731	0.238	0.517	618	7268
nationality	0.555	0.171	0.383	386	7036
parents	0.862	0.544	0.771	390	7040
place_of_birth	0.637	0.206	0.449	282	6932
place_of_death	0.512	0.100	0.282	209	6859
profession	0.716	0.205	0.477	308	6958
worked_at	0.688	0.254	0.513	303	6953
macro-average	0.732	0.328	**0.567**	11210	117610

A simple classifier: running experiments

```
_ = rel_ext.experiment(
    splits,
    featurizers=[simple_bag_of_words_featurizer])
```

relation	precision	recall	f-score	support	size
adjoins	0.832	0.378	0.671	407	7057
author	0.779	0.525	0.710	657	7307
capital	0.638	0.294	0.517	126	6776
contains	0.783	0.608	0.740	4487	11137
film_performance	0.796	0.591	0.745	984	7634
founders	0.783	0.384	0.648	469	7119
genre	0.654	0.166	0.412	205	6855
has_sibling	0.865	0.246	0.576	625	7275
has_spouse	0.878	0.342	0.668	754	7404
is_a	0.731	0.238	0.517	618	7268
nationality	0.555	0.171	0.383	386	7036
parents	0.862	0.544	0.771	390	7040
place_of_birth	0.637	0.206	0.449	282	6932
place_of_death	0.512	0.100	0.282	209	6859
profession	0.716	0.205	0.477	308	6958
worked_at	0.688	0.254	0.513	303	6953
macro-average	0.732	0.328	**0.567**	11210	117610