Relation Extraction

Presented
By
Chandresh Kumar Maurya
IIT Indore

Slides adapted from Bill MacCartney leccture on relation extraction

Overview

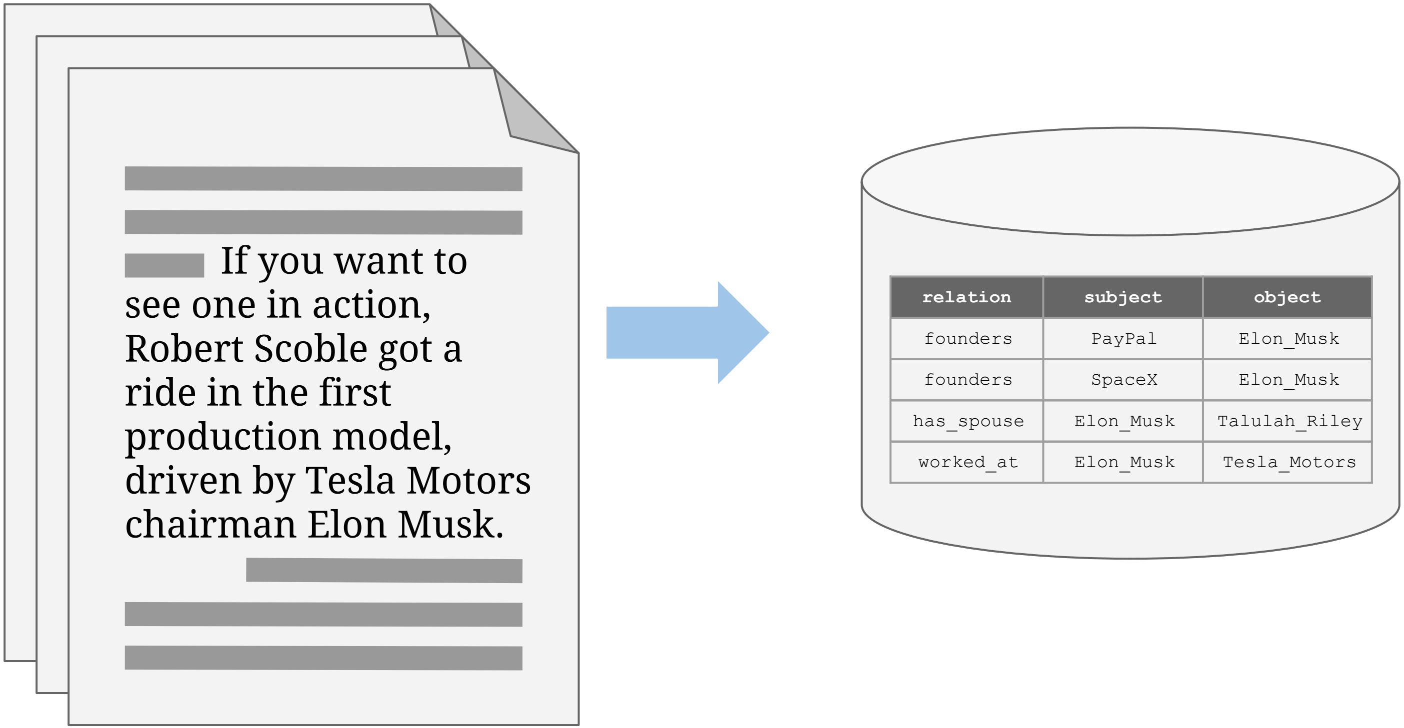
- The task of relation extraction
- Data resources
- Problem formulation
- Evaluation
- Simple baselines
- Directions to explore

The task of relation extraction

- Task definition
- Goal: machine reading
- Practical applications
- Hand-built patterns
- Supervised learning
- Distant supervision

The task of relation extraction

Task definition



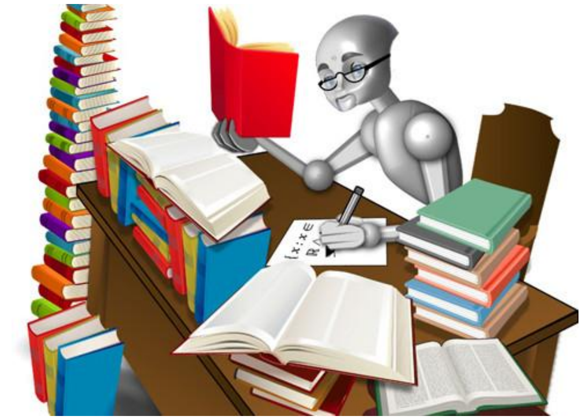
relation	subject	object
founders	PayPal	Elon_Musk
founders	SpaceX	Elon_Musk
has_spouse	Elon_Musk	Talulah_Riley
worked_at	Elon_Musk	Tesla_Motors

Goal: machine reading

Reading the Web: A Breakthrough Goal for AI

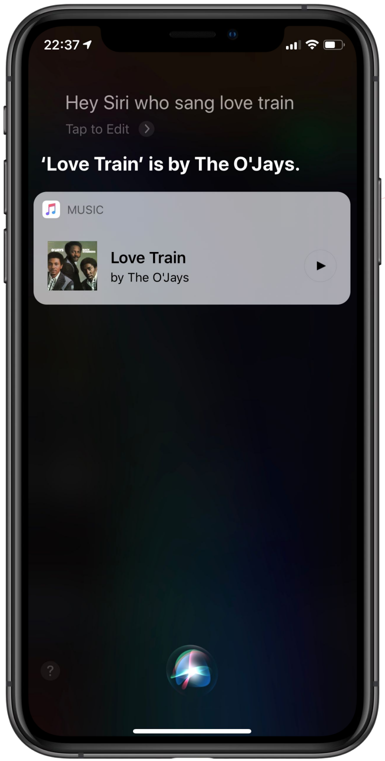
I believe AI has an opportunity to achieve a true breakthrough over the coming decade by at last solving the problem of reading natural language text to extract its factual content. In fact, I hereby offer to bet anyone a lobster dinner that by 2015 we will have a computer program capable of automatically reading at least 80% of the factual content [on the] web, and placing those facts in a structured knowledge base. The significance of this AI achievement would be tremendous: it would immediately increase by many orders of magnitude the volume, breadth, and depth of ground facts and general knowledge accessible to knowledge based AI programs. In essence, computers would be harvesting in structured form the huge volume of knowledge that millions of humans are entering daily on the web in the form of unstructured text.

— Tom Mitchell, 2005



The task of relation extraction

Applications: intelligent assistants



/music/artist/track

The O'Jays	Love Train
Cardi B	Bodak Yellow
Selena Gomez	Bad Liar

/film/film/starring

Wonder Woman	Gal Gadot
Dunkirk	Tom Hardy
Tomb Raider	Alicia Vikander

/organization/organization/parent

tbh	Facebook
Kaggle	Google
LinkedIn	Microsoft

/people/person/date_of_death

Barbara Bush	2018-04-17
Milos Forman	2018-04-14
Winnie Mandela	2018-04-11

"Love Train" is a hit single by The O'Jays, written by Kenny Gamble and Leon Huff. Released in 1972, it reached number one on both the R&B Singles and the Billboard Hot 100, in February and March 1973 respectively, number 9 on the UK Singles Chart and was certified gold by the RIAA. It was The O'Jays' first and only number-one record on the US pop chart.

Applications: building ontologies

```
video game
  action game
    ball and paddle game
      Breakout
    platform game
      Donkey Kong
    shooter
      arcade shooter
        Space Invaders
      first-person shooter
        Call of Duty
      third-person shooter
        Tomb Raider
  adventure game
    text adventure
    graphic adventure
  strategy game
    4X game
      Civilization
    tower defense
      Plants vs. Zombies
```

Mirror ran a headline questioning whether the killer's actions were a result of playing Call of Duty, a first-person shooter game …

Melee, in video game terms, is a style of elbow-drop hand-to-hand combat popular in first-person shooters and other shooters.

Tower defense is a kind of real-time strategy game in which the goal is to protect an area or place and prevent enemies from reaching …

The task of relation extraction

Applications: gene regulation


Involvement of Tumor Necrosis Factor Receptor-associated Protein 1 (TRAP1) in Apoptosis Induced by β-Hydroxyisovalerylshikonin

relation	subject	object
is_a	p53	protein
is_a	Bax	protein
has_function	p53	apoptosis
has_function	Bax	induction
involved_in	apoptosis	cell_death
is_in	Bax	cytoplasm
related_to	apoptosis	caspase_activation
...	...	...

textual abstract:
summary for human

structured knowledge extraction:
summary for machine

Hand-built patterns

Idea: define some extraction patterns

X is the founder of Y	→	48-year-old Elon Musk is the founder of SpaceX and a co-founder of Tesla Motors.
X, who founded Y	→	Elon Musk, who founded SpaceX in 2002, has said the company is focused on ...
Y was founded by X	→	SpaceX was founded by Elon Musk to make life multi-planetary. “You want to ...

Problem: most occurrences do not fit simple patterns

- You may also be thinking of Elon Musk (founder of SpaceX), who started PayPal.
- Elon Musk, co-founder of PayPal, went on to establish SpaceX, one of the most ...
- If Space Exploration (SpaceX), founded by Paypal pioneer Elon Musk succeeds, ...

The task of relation extraction

Supervised learning

Idea: label examples, train a classifier

You may also be thinking of Elon Musk (founder of SpaceX), who started PayPal.	✓
Elon Musk, co-founder of PayPal, went on to establish SpaceX, one of the most …	✓
If Space Exploration (SpaceX), founded by Paypal pioneer Elon Musk succeeds, …	✓
Entrepreneur Elon Musk announced the latest addition to the SpaceX arsenal …	✗
Elon Musk tweeted Friday that SpaceX employees are “working on ventilators” …	✗

founder	+1.56
founded	+1.41
establish	+1.23
co-founder	+1.01
PayPal	+0.35
announced	−0.23
addition	−0.32
tweeted	−0.44

Success! Better generalizability

Problem: labeling examples is expensive :-(

Distant supervision

Idea: derive labels from an existing knowledge base (KB)

Assume sentences with related entities are positive examples

Assume sentences with unrelated entities are negative examples

relation	subject	object
founders	SpaceX	Elon_Musk
founders	Apple	Steve_Jobs
founders	Microsoft	Bill_Gates

Elon Musk, co-founder of PayPal, went on to establish SpaceX, one of the most ...	✓
Entrepreneur Elon Musk announced the latest addition to the SpaceX arsenal ...	✓
Elon Musk dismissed concerns that Apple was poaching the company's talent.	✗
Now we know what Apple would have done with Elon Musk if that deal had ...	✗

Hooray! Massive quantities of training data, practically free!

Qualm: are those assumptions reliable?

Distant supervision: limitations

Distant supervision is a powerful idea — but it has two limitations:

1. Not all sentences with related entities are truly positive examples

    Entrepreneur Elon Musk announced the latest addition to the SpaceX arsenal ...

    (but the benefit of *more* data outweighs the harm of noisier data)

2. Need an existing KB to start from — can't start from scratch

Overview

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

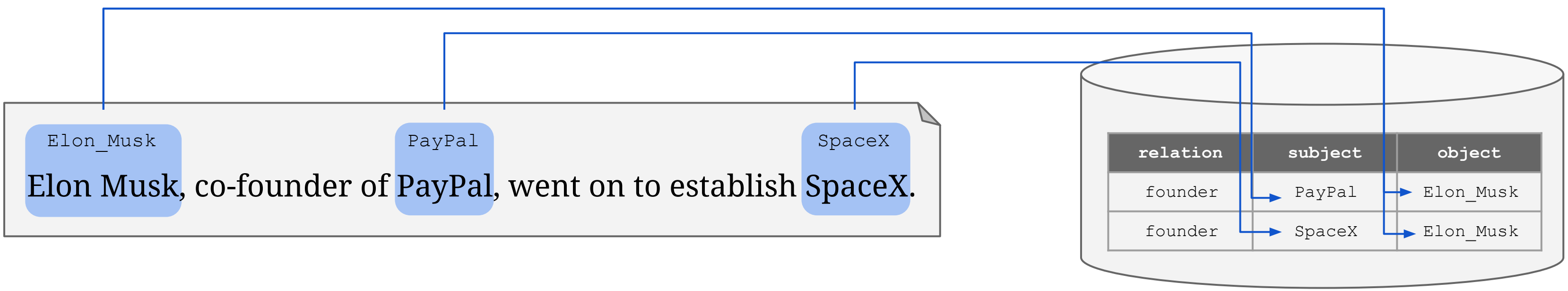
Data resources

- The corpus
- The knowledge base (KB)

Data resources

The corpus

We need a corpus of sentences, each containing a pair of entities which have been annotated with entity resolutions so that they can be unambiguously linked to a knowledge base



Solution: the Wikilinks corpus (heavily adapted for our purposes)

The corpus: the `Corpus` class

The `Corpus` class holds examples, and allows lookup by entity:

```
rel_ext_data_home = os.path.join('data', 'rel_ext_data')
corpus = rel_ext.Corpus(os.path.join(rel_ext_data_home,'corpus.tsv.gz'))
print('Read {0:,} examples'.format(len(corpus)))
```

```
Read 331,696 examples
```

```
print(corpus.examples[1])
```

```
Example(entity_1='New_Mexico', entity_2='Arizona', left='to all Spanish-occupied lands . The horno has a
beehive shape and uses wood as the only heat source . The procedure still used in parts of', mention_1='New
Mexico', middle='and', mention_2='Arizona', right='is to build a fire inside the Horno and , when the proper
amount of time has passed , remove the embers and ashes and insert the', left_POS='to/TO all/DT
Spanish-occupied/JJ lands/NNS ./. The/DT horno/NN has/VBZ a/DT beehive/NN ... ')
```

Data resources

The corpus: the `Example` class


```
Example = namedtuple('Example',
    'entity_1, entity_2, left, mention_1, middle, mention_2, right, '
    'left_POS, mention_1_POS, middle_POS, mention_2_POS, right_POS')
```

	`New_Mexico`		`Arizona`	
	`entity_1`		`entity_2`	
The procedure still used in parts of	New Mexico	and	Arizona	is to build a fire inside the Horno ...
`left`	`mention_1`	`middle`	`mention_2`	`right`
The/DT procedure/NN still/RB used/VBN in/IN parts/NNS of/IN	New/NNP Mexico/NNP	and/CC	Arizona/NNP	is/VBZ to/TO build/VB a/DT fire/NN inside/IN the/DT Horno/NNP ...
`left_POS`	`mention_1_POS`	`middle_POS`	`mention_2_POS`	`right_POS`

The corpus: most common entities

```
counter = Counter()
for example in corpus.examples:
    counter[example.entity_1] += 1
    counter[example.entity_2] += 1
print('The corpus contains {} entities'.format(len(counter)))
counts = sorted([(count, key) for key, count in counter.items()], reverse=True)
print('The most common entities are:')
for count, key in counts[:10]:
    print('{:10d} {}'.format(count, key))
```

```
The corpus contains 95909 entities
The most common entities are:
      8137 India
      5240 England
      4121 France
      4040 Germany
      3937 Australia
      3779 Canada
      3633 Italy
      3138 California
      2894 New_York_City
      2745 Pakistan
```

The corpus: finding examples by entities

```
corpus.show_examples_for_pair('Elon_Musk', 'Tesla_Motors')
```

```
The first of 5 examples for Elon Musk and Tesla Motors is:
Example(entity 1='Elon Musk', entity 2='Tesla Motors', left='space for a while , here ’ s what might be
launching Americans into space in the next decade . Falcon 9 From sometimes Canadian , South African &
American', mention 1='Elon Musk', middle='‘ s company Space X . Musk is a PayPal alumni and',
mention 2='Tesla Motors', right='co-founder - remember that latter company name for future trivia questions
and/or a remake of Back to the Future . After several successful launches on their Falcon',...)
```

```
corpus.show_examples_for_pair('Tesla_Motors', 'Elon_Musk')
```

```
The first of 2 examples for Tesla Motors and Elon Musk is:
Example(entity 1='Tesla Motors', entity 2='Elon Musk', left='their factory in Hethel . If you want to see
one in action , Robert Scoble got a ride in the first production model , driven by', mention 1='Tesla
Motors', middle='chairman', mention 2='Elon Musk', right='. Needless to say he got the whole thing on video
, and covers a lot of technical details about the car - this is the',...)
```

The corpus: final observations

The Wikilinks corpus has some flaws. For example, it contains many near-dupes — an artefact of the document sampling methodology used to construct it.

One thing this corpus does *not* include is any annotation about relations. So, can’t be used for the fully-supervised approach.

To make headway, we need to connect the corpus to a KB!

Data resources

The knowledge base (KB)

Our KB is derived from Freebase (which shut down in 2016 😞).

It contains relational triples of the form (relation, subject, object).

```
(place_of_birth, Barack_Obama, Honolulu)
(has_spouse, Barack_Obama, Michelle_Obama)
(author, The_Audacity_of_Hope, Barack_Obama)
```

The relation is one of a handful of predefined constants.

The subject and object are entities identified by Wiki IDs.

The knowledge base: the `KB` class

The `KB` class holds `KBTriples`, and allows lookup by entity:

```
kb = rel_ext.KB(os.path.join(rel_ext_data_home,'kb.tsv.gz'))

print('Read {0:,} KB triples'.format(len(kb)))
```

```
Read 45,884 KB triples
```

```
print(kb.kb_triples[0])
```

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

The knowledge base: data exploration

```
len(kb.all_relations)
```

```
16
```

The knowledge base: data exploration

```
for rel in kb.all_relations:
    print('{:12d} {}'.format(len(kb.get_triples_for_relation(rel)), rel))
```

```
        1702 adjoins
        2671 author
         522 capital
       18681 contains
        3947 film_performance
        1960 founders
         824 genre
        2563 has_sibling
        2994 has_spouse
        2542 is_a
        1598 nationality
        1586 parents
        1097 place_of_birth
         831 place_of_death
        1216 profession
        1150 worked_at
```

The knowledge base: data exploration

```
for rel in kb.all_relations:
    print(tuple(kb.get_triples_for_relation(rel)[0]))
```

```
('adjoins', 'France', 'Spain')
('author', 'Uncle_Silas', 'Sheridan_Le_Fanu')
('capital', 'Panama', 'Panama_City')
('contains', 'Brickfields', 'Kuala_Lumpur_Sentral_railway_station')
('film_performance', 'Colin_Hanks', 'The_Great_Buck_Howard')
('founders', 'Lashkar-e-Taiba', 'Hafiz_Muhammad_Saeed')
('genre', '8_Simple_Rules', 'Sitcom')
('has_sibling', 'Ari_Emanuel', 'Rahm_Emanuel')
('has_spouse', 'Percy_Bysshe_Shelley', 'Mary_Shelley')
('is_a', 'Bhanu_Athaiya', 'Costume_designer')
('nationality', 'Ruben_Rausing', 'Sweden')
('parents', 'Rosanna_Davison', 'Chris_de_Burgh')
('place_of_birth', 'William_Penny_Brookes', 'Much_Wenlock')
('place_of_death', 'Jean_Drapeau', 'Montreal')
('profession', 'Rufus_Wainwright', 'Actor')
('worked_at', 'Brian_Greene', 'Columbia_University')
```

The knowledge base: data exploration

The `get_triples_for_entities()` method allows easy lookup:

```
kb.get_triples_for_entities('France', 'Germany')
```

```
[KBTriple(rel='adjoins', sbj='France', obj='Germany')]
```

```
kb.get_triples_for_entities('Germany', 'France')
```

```
[KBTriple(rel='adjoins', sbj='Germany', obj='France')]
```

Relations like `adjoins` are intuitively symmetric — but there's no guarantee that such inverse triples actually appear in the KB!

The knowledge base: data exploration

Most relations are intuitively asymmetric:

```
kb.get_triples_for_entities('Tesla_Motors', 'Elon_Musk')
```

```
[KBTriple(rel='founders', sbj='Tesla_Motors', obj='Elon_Musk')]
```

```
kb.get_triples_for_entities('Elon_Musk', 'Tesla_Motors')
```

```
[KBTriple(rel='worked_at', sbj='Elon_Musk', obj='Tesla_Motors')]
```

So it can be the case that one relation holds between *X* and *Y*, and a different relation holds between *Y* and *X*.

The knowledge base: data exploration

An entity pair can belong to multiple relations.

```
kb.get_triples_for_entities('Cleopatra', 'Ptolemy_XIII_Theos_Philopator')
```

```
[KBTriple(rel='has_sibling', sbj='Cleopatra', obj='Ptolemy_XIII_Theos_Philopator'),
 KBTriple(rel='has_spouse', sbj='Cleopatra', obj='Ptolemy_XIII_Theos_Philopator')]
```

The knowledge base: data exploration

```
counter = Counter()
for kbt in kb.kb_triples:
    counter[kbt.sbj] += 1
    counter[kbt.obj] += 1
print('The KB contains {:,} entities'.format(len(counter)))
counts = sorted([(count, key) for key, count in counter.items()], reverse=True)
print('The most common entities are:')
for count, key in counts[:10]:
    print('{:10d} {}'.format(count, key))
```

```
The KB contains 40,141 entities
The most common entities are:
       945 England
       786 India
       438 Italy
       414 France
       412 California
       400 Germany
       372 United_Kingdom
       366 Canada
       302 New_York_City
       247 New_York
```

The knowledge base: data exploration

Note, no promise or expectation that the KB is *complete!*

In the KB:

```
(founders, Tesla_Motors, Elon_Musk)
(worked_at, Elon_Musk, Tesla_Motors)
(founders, SpaceX, Elon_Musk)
```

Not in the KB:

```
(worked_at, Elon_Musk, SpaceX)
```

Overview

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

Problem formulation

- Inputs and outputs
- Joining the corpus and the KB
- Negative instances
- Multi-label classification

Inputs and outputs

What is the input to the prediction?

- A pair of entity mentions in the context of a sentence?
- A pair of entities, independent of any specific context?

What is the output to the prediction?

- A single relation (multi-class classification)?
- Or multiple relations (multi-label classification)?

Problem formulation

Joining the corpus and the KB

Classifying a pair of entity mentions in corpus? Get labels from KB.

Elon Musk, co-founder of PayPal, went on to establish SpaceX, ...

✓

relation	subject	object
founder	SpaceX	Elon_Musk

Classifying a pair of entities for the KB? Get features from corpus.

You may also be thinking of Elon Musk (founder of SpaceX), who ...

Elon Musk announced the latest addition to the SpaceX arsenal ...

If Space Exploration (SpaceX), founded by Paypal pioneer Elon Musk ...

1	addition
1	announced
1	by
1	founded
1	founder
1	latest
1	of
1	PayPal
1	pioneer
2	the
1	to

(Elon_Musk, SpaceX)

Problem formulation

Joining the corpus and the KB

```
dataset = rel_ext.Dataset(corpus, kb)
dataset.count_examples()
```

relation	examples	triples	examples /triple
adjoins	58854	1702	34.58
author	11768	2671	4.41
capital	7443	522	14.26
contains	75952	18681	4.07
film_performance	8994	3947	2.28
founders	5846	1960	2.98
genre	1576	824	1.91
has_sibling	8525	2563	3.33
has_spouse	12013	2994	4.01
is_a	5112	2542	2.01
nationality	3403	1598	2.13
parents	3802	1586	2.40
place_of_birth	1657	1097	1.51
place_of_death	1523	831	1.83
profession	1851	1216	1.52
worked_at	3226	1150	2.81

Negative instances

To train a classifier, we also need negative instances!

So, find corpus examples containing pairs of entities not related in KB

```
unrelated_pairs = dataset.find_unrelated_pairs()
print('Found {0:,} unrelated pairs, including:'.format(len(unrelated_pairs)))
for pair in list(unrelated_pairs)[:10]:
    print('    ', pair)
```

```
Found 247,405 unrelated pairs, including:
     ('Inglourious_Basterds', 'Christoph_Waltz')
     ('NBCUniversal', 'E!')
     ('The_Beatles', 'Keith_Moon')
     ('Patrick_Lussier', 'Nicolas_Cage')
     ('Townes_Van_Zandt', 'Johnny_Cash')
     ('UAE', 'Italy')
     ('Arshile_Gorky', 'Hans_Hofmann')
     ('Sandra_Bullock', 'Jae_Head')
```

Problem formulation

Multi-label classification

Many entity pairs belong to more than one relation:

```
dataset.count_relation_combinations()
```

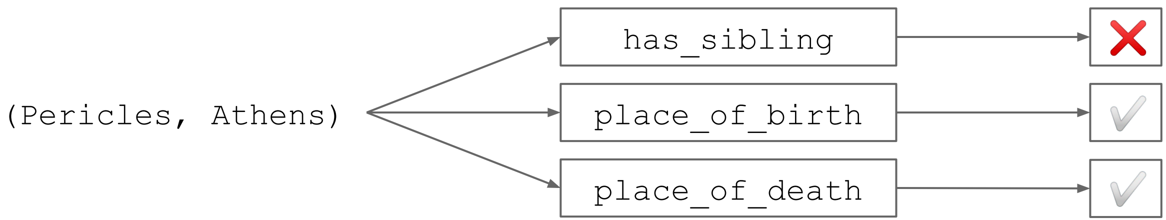
```
The most common relation combinations are:
      1216 ('is_a', 'profession')
       403 ('capital', 'contains')
       143 ('place_of_birth', 'place_of_death')
        61 ('nationality', 'place_of_birth')
        11 ('adjoins', 'contains')
         9 ('nationality', 'place_of_death')
         7 ('has_sibling', 'has_spouse')
         3 ('nationality', 'place_of_birth', 'place_of_death')
         2 ('parents', 'worked_at')
```

This suggests formulating our problem as *multi-label classification*.

Multi-label classification: binary relevance

Many possible approaches to multi-label classification.

The most obvious is the *binary relevance method:*
just train a separate binary classifier for each label.



Disadvantage: fails to exploit correlations between labels.

Advantage: simple.

Binary classification of KB triples

So here's the problem formulation we've arrived at:

Input:	an entity pair and a candidate relation
Output:	does the entity pair belong to the relation?

In other words: binary classification of KB triples!

That is, given a candidate KB triple, do we predict that it is valid?

```
(worked_at, Elon_Musk, SpaceX) ?
```

Overview

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

Evaluation

- Test-driven development
- Splitting the data
- Precision and recall
- F-measure
- Micro-averaging and macro-averaging
- Figure of merit

Test-driven development

Good software engineering uses *test-driven development*:

First, write unit tests that check whether the code works.

Then, start writing the code, iterating until it passes the tests.

Good model engineering can use a similar paradigm:

First, build a test harness that performs a quantitative evaluation.

Then, start building models, hill-climbing on your evaluation.

Splitting the data

As usual, we'll want to partition our data into multiple splits:

Tiny	1%
Train	74%
Dev	25%
Test	?

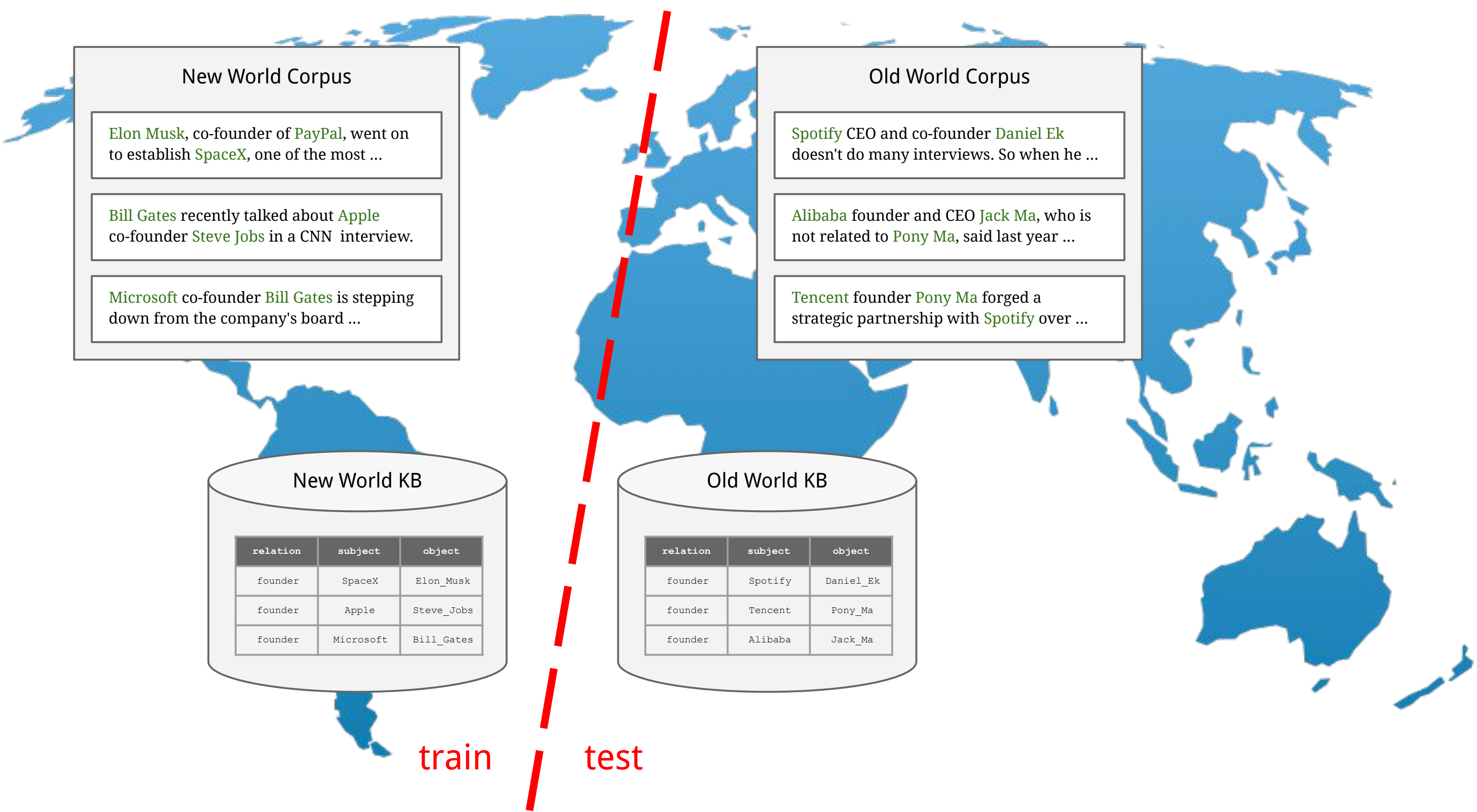
Complication: we need to split both corpus and KB.

We want relations to span splits, so that we can assess our success in learning how a given relation is expressed in natural language.

But ideally, we'd like the splits to *partition* the entities, to avoid leaks.

Evaluation

Splitting the data: the ideal



Splitting the data: the achievable

But the world is strongly entangled, and the ideal is hard to achieve.

Instead, we'll approximate the ideal:

- First, split KB triples by subject entity.
- Then, split corpus examples:
  - If entity_1 is in a split, assign example to that split.
  - Or, if entity_2 is in a split, assign example to that split.
  - Otherwise, assign example to split randomly.

Splitting the data: `build_splits()`

```
splits = dataset.build_splits(
    split_names=['tiny', 'train', 'dev'],
    split_fracs=[0.01, 0.74, 0.25],
    seed=1)

splits
```

```
{'tiny': Corpus with 3,474 examples; KB with 445 triples,
 'train': Corpus with 249,003 examples; KB with 34,229 triples,
 'dev': Corpus with 79,219 examples; KB with 11,210 triples,
 'all': Corpus with 331,696 examples; KB with 45,884 triples}
```

Evaluation

Precision and recall

Precision and recall are the standard metrics for binary classification.

		predicted		
		F	T	
actual	F	86	2	88
	T	9	3	12
		95	5	100

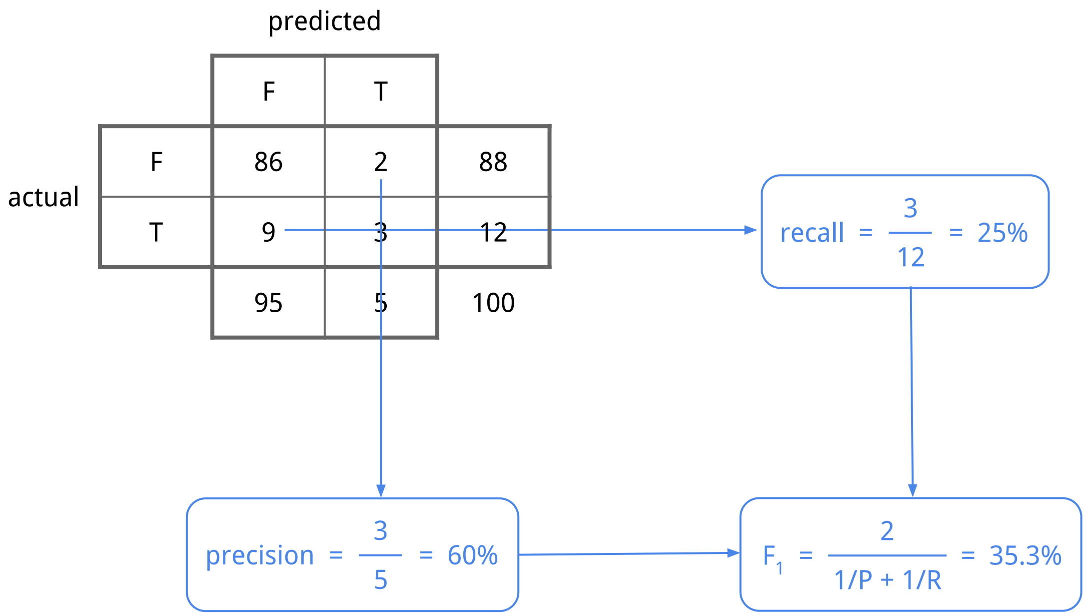
$$\text{recall} = \frac{3}{12} = 25\%$$

$$\text{accuracy} = \frac{89}{100} = 89\%$$

$$\text{precision} = \frac{3}{5} = 60\%$$

F_1

The F_1 score combines precision and recall using the harmonic mean.



Evaluation

F-measure

F-measure is a weighted combination of precision and recall.

$$F_\beta = \frac{1 + \beta^2}{1/P + \beta^2/R}$$

P	0.800	high precision
R	0.200	low recall
F_1	0.320	equal weight to precision and recall
$F_{0.5}$	0.500	more weight to precision
F_2	0.235	more weight to recall

For relation extraction, precision probably matters more than recall.

So, let's use $F_{0.5}$ as our evaluation metric.

Micro-averaging and macro-averaging

Micro-averaging gives equal weight to each problem instance.

Macro-averaging gives equal weight to each relation.

relation	instances	F-score
adjoins	100	0.700
author	100	0.800
contains	1000	0.900
micro-average		0.875
macro-average		0.800

We'll use macro-averaging, so that we don't overweight large relations.

Figure of merit

Your “figure of merit” is the one metric — a *single* number — you’re seeking to optimize in your iterative development process.

We’re choosing macro-averaged $F_{0.5}$ as our figure of merit.

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.

Simple baselines

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
```

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
```

Simple baselines

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')))
```

Simple baselines

A simple classifier: making predictions

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

Simple baselines

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

Simple baselines

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

Overview

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

Directions to explore

- Examining the trained models
- Discovering new relation instances
- Enhancing the model

Directions to explore

Examining the trained models

```
rel_ext.examine_model_weights(train_result)
```

```
Highest and lowest feature weights for relation author:

     3.055 author
     3.032 books
     2.342 by
     ..... .....
    -2.002 directed
    -2.019 or
    -2.211 poetry
```

```
Highest and lowest feature weights for relation adjoins:

     2.511 Córdoba
     2.467 Taluks
     2.434 Valais
     ..... .....
    -1.143 for
    -1.186 Egypt
    -1.277 America
```

```
Highest and lowest feature weights for relation
film_performance:

     4.004 starring
     3.731 alongside
     3.199 opposite
     ..... .....
    -1.702 then
    -1.840 She
    -1.889 Genghis
```

```
Highest and lowest feature weights for relation has_spouse:

     5.319 wife
     4.652 married
     4.617 husband
     ..... .....
    -1.528 between
    -1.559 MTV
    -1.599 Terri
```

Directions to explore

Discovering new relation instances

```
rel_ext.find_new_relation_instances(
    dataset,
    featurizers=[simple_bag_of_words_featurizer])
```

```
Highest probability examples for relation adjoins:

     1.000 KBTriple(rel='adjoins', sbj='Canada', obj='Vancouver')
     1.000 KBTriple(rel='adjoins', sbj='Vancouver', obj='Canada')
     1.000 KBTriple(rel='adjoins', sbj='Australia', obj='Sydney')
     1.000 KBTriple(rel='adjoins', sbj='Sydney', obj='Australia')
     1.000 KBTriple(rel='adjoins', sbj='Mexico', obj='Atlantic_Ocean')
     1.000 KBTriple(rel='adjoins', sbj='Atlantic_Ocean', obj='Mexico')
     1.000 KBTriple(rel='adjoins', sbj='Dubai', obj='United_Arab_Emirates')
     1.000 KBTriple(rel='adjoins', sbj='United_Arab_Emirates', obj='Dubai')
     1.000 KBTriple(rel='adjoins', sbj='Sydney', obj='New_South_Wales')
     1.000 KBTriple(rel='adjoins', sbj='New_South_Wales', obj='Sydney')
```

Directions to explore

Discovering new relation instances

```
rel_ext.find_new_relation_instances(
    dataset,
    featurizers=[simple_bag_of_words_featurizer])
```

```
Highest probability examples for relation author:

     1.000 KBTriple(rel='author', sbj='Oliver_Twist', obj='Charles_Dickens')
     1.000 KBTriple(rel='author', sbj='Jane_Austen', obj='Pride_and_Prejudice')
     1.000 KBTriple(rel='author', sbj='Iliad', obj='Homer')
     1.000 KBTriple(rel='author', sbj='Divine_Comedy', obj='Dante_Alighieri')
     1.000 KBTriple(rel='author', sbj='Pride_and_Prejudice', obj='Jane_Austen')
     1.000 KBTriple(rel='author', sbj="Euclid's_Elements", obj='Euclid')
     1.000 KBTriple(rel='author', sbj='Aldous_Huxley', obj='The_Doors_of_Perception')
     1.000 KBTriple(rel='author', sbj="Uncle_Tom's_Cabin", obj='Harriet_Beecher_Stowe')
     1.000 KBTriple(rel='author', sbj='Ray_Bradbury', obj='Fahrenheit_451')
     1.000 KBTriple(rel='author', sbj='A_Christmas_Carol', obj='Charles_Dickens')
```

Directions to explore

Discovering new relation instances

```
rel_ext.find_new_relation_instances(
    dataset,
    featurizers=[simple_bag_of_words_featurizer])
```

```
Highest probability examples for relation capital:

     1.000 KBTriple(rel='capital', sbj='Delhi', obj='India')
     1.000 KBTriple(rel='capital', sbj='Bangladesh', obj='Dhaka')
     1.000 KBTriple(rel='capital', sbj='India', obj='Delhi')
     1.000 KBTriple(rel='capital', sbj='Lucknow', obj='Uttar_Pradesh')
     1.000 KBTriple(rel='capital', sbj='Chengdu', obj='Sichuan')
     1.000 KBTriple(rel='capital', sbj='Dhaka', obj='Bangladesh')
     1.000 KBTriple(rel='capital', sbj='Uttar_Pradesh', obj='Lucknow')
     1.000 KBTriple(rel='capital', sbj='Sichuan', obj='Chengdu')
     1.000 KBTriple(rel='capital', sbj='Bandung', obj='West_Java')
     1.000 KBTriple(rel='capital', sbj='West_Java', obj='Bandung')
```

Directions to explore

Discovering new relation instances

```
rel_ext.find_new_relation_instances(
    dataset,
    featurizers=[simple_bag_of_words_featurizer])
```

```
Highest probability examples for relation worked_at:

     1.000 KBTriple(rel='worked_at', sbj='William_C._Durant', obj='Louis_Chevrolet')
     1.000 KBTriple(rel='worked_at', sbj='Louis_Chevrolet', obj='William_C._Durant')
     1.000 KBTriple(rel='worked_at', sbj='Iliad', obj='Homer')
     1.000 KBTriple(rel='worked_at', sbj='Homer', obj='Iliad')
     1.000 KBTriple(rel='worked_at', sbj='Marvel_Comics', obj='Stan_Lee')
     1.000 KBTriple(rel='worked_at', sbj='Stan_Lee', obj='Marvel_Comics')
     1.000 KBTriple(rel='worked_at', sbj='Mongol_Empire', obj='Genghis_Khan')
     1.000 KBTriple(rel='worked_at', sbj='Genghis_Khan', obj='Mongol_Empire')
     1.000 KBTriple(rel='worked_at', sbj='Comic_book', obj='Marvel_Comics')
     1.000 KBTriple(rel='worked_at', sbj='Marvel_Comics', obj='Comic_book')
```

Directions to explore

Error analysis

```
exs = dataset.corpus.get_examples_for_entities( 'Louis_Chevrolet' , 'William_C._Durant' )
for ex in exs:
    print(' | '.join((ex.left[ -10:], ex.mention_1, ex.middle, ex.mention_2, ex.right[:  10])))
```

```
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
Founded by | Louis Chevrolet | and ousted GM founder | William C. Durant | on Novembe
```

```
model = train_result[ 'models' ]['worked_at' ]
vectorizer = train_result[ 'vectorizer' ]
print(model.coef_[0][vectorizer.vocabulary_[  'founder' ]])
```

```
2.0528435038145383
```

Directions to explore

Error analysis

```
print(len(dataset.corpus.get_examples_for_entities( 'Homer', 'Iliad')))
```

```
118
```

```
mids = defaultdict(int)
for ex in dataset.corpus.get_examples_for_entities( 'Homer', 'Iliad'):
    mids[ex.middle]  += 1
for cnt, mid in sorted([(cnt, mid)  for mid, cnt in mids.items()], reverse =True)[:5]:
    print('{:10d} {:s}' .format(cnt, mid))
```

```
        51 's
        13 ` s
         4 , and in particular the
         4 ,
         3 in the
```

```
model = train_result['models']['worked_at']
vectorizer = train_result['vectorizer']
print(model.coef_[0][vectorizer.vocabulary_[ "'s"]])
```

```
0.5801433006163413
```

Enhancing the model: feature representations

- Word embeddings
- Directional bag-of-words
- N-grams
- POS tags
- WordNet synsets
- Syntactic features
- Features based on entity mentions
- Features based on `left` and `right`

Directions to explore

Enhancing the model: model types

- Support vector machines (SVMs)
- Feed-forward neural networks
- LSTMs
- Transformers