137: Named Entity Recognition

Aaron Levine, Tianzhi Chen, Zachary Yocum

Task

- BIO tag labels
    - `[B-PER, O, O, B-LOC, I-LOC, O, ...]`

- Only B and I evaluated
    - NER classes ignored
    - recall, precision and f1 measure

Dataset

Issue	128 docs instead of 1000s	90% of data is tagged with “O”
Result	lots of potential words we’ve *never seen*	any overlapping features between B/I and O will be heavily weighted towards O
Solutions	simplify data: ● clustering ● POS tags ● compare against gazetteer ● stemming ● etc.	get more context: ● bigrams ● trigrams ● compare against named entity list ● prev / next token feature sets ● etc.

CRFSuite

- A fast implementation of Conditional Random Fields (CRFs)
  - Model optimization using limited memory **B**royden-**F**letcher-**G**oldfarb-**S**hanno (L-BFGS)
  - Claims to be 5.4-61.8 x faster than CRF++
- Simple, white-space separated feature template
- Supports model inspection
  - CRF model binaries can be dumped to human-readable text to view feature weights

Brown Cluster

Python implementation of TAN clustering

- input: newline separated sentences culled from data
- output: newline separated tuples of:
  - token
  - binary tree index label
  - cluster label
- system uses both training and test data, as it's not accessing the BIO tags
- 132 tags on our set

word2vec

Google's description:

"The `word2vec` *tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words. The resulting word vector file can be used as features in many natural language processing and machine learning applications."*

word2vec

- Uses a vector-space model to map words to points in a high-dimensional space
    - *N*-dimensional space where *N*=|vocabulary|
- Provides word similarity quantification
    - Measures cos(θ) where θ is the angle between word vectors
- Allows for clustering with `word2clusters`
    - K-means clustering on top of word vectors based on cos (θ)
    - we matched our Brown cluster partition with 132 clusters

Gazetteer

CoNLL 2003 Named Entity Recognition competition

- list of 8215 named entities and entity types:

```
LOC Asia
LOC ASIA
LOC ASIA PACIFIC
LOC Asmara
...
```

- case sensitive: Asia and ASIA both in set
  - avoids classifying “us” (pronoun) as named entity “US” (country)

Gazetteer

```
gazetteer['Khan'] →
    [(1, ['Gujar', 'Khan'], 'LOC'),
     (1, ['Faheem', 'Khan'], 'PER'),
     (1, ['Imran', 'Khan'], 'PER'),
     (1, ['Jansher', 'Khan'], 'PER'),
     (0, ['Khan'], 'PER'),
     (1, ['Moin', 'Khan'], 'PER'),
     (2, ['Shahid', 'Hasan', 'Khan'], 'PER'),
     (1, ['W.', 'Khan'], 'PER'),
     (2, ['Zarak', 'Jahan', 'Khan'], 'PER'),
     (2, ['Zubair', 'Jahan', 'Khan'], 'PER')]
```

Gazetteer Lookup

For each token in sentence:

- key into list of named entities with that token
- for each potential entity:
    - use entity token count and target token index to pull surrounding tokens in sentence
    - if tokens in sentence and tokens in entity match, return the entity type as a string
    - else, return `None`

Gazetteer Lookup

Create dictionary keyed on words:

- values are list of all named entities containing key token as tuple of:
    - tokenized entity
    - position of token in tokenized entity
    - entity tag
    - ex: `“United”` → `[(“United States”, 1, “LOC”),`

```
(“United Farm Workers”, 1,“ORG”),
...]
```

14 Unigram Features

Name	Type	Description
cap	bool	is the entire token string capitalized?
title	bool	is the entire token string titlecase?
alnum	bool	is the entire token string alphanumeric
num	bool	is the entire token string numeric?
alpha	bool	is the entire token string alphabetic?
first	bool	is the token the first token of the sentence?
nopunct	str	token string without leading and trailing punctuation
shape	str	string representing the orthographical class of the token
simple_shape	str	simple orthographical representation (Xxx, xxxxx, xXxxx, etc.)
pos	str	part-of-speech (POS) tag of the token
entity_type	str	NER class via gazetteer lookup (GPE, PER, FAC, etc.)
length	int	count of the characters in the token string
brown_cluster_id	int	Brown cluster ID looked up from a dictionary
w2vcluster	int	word2vec cluster ID looked up from a dictionary

6 Bi/Trigram Features

Name	**Type**	**Description**
prev_bigram_pos	[str,str]	pos[i-1,i]
prev_bigram	[str,str]	word[i-1,i]
trigram_pos	[str,str,str]	pos[i-1,i,i+1]
trigram	[str,str,str]	word[i-1,i,i+1]
next_trigram	[str,str,str]	word[i,i+1,i+2]
prev_trigram	[str,str,str]	word[i-2,i-1,i]

Sequential Context Features

Total of 60 features

- 20 feature functions x 3 context windows
- Experimented with various context windows
  - previous token and target token
  - target token and next token
  - target token and previous two tokens
  - etc.
- Larger contexts hurt performance
- Ultimately settled on `[i-1, i, i+1]`

N-Gram Features

- Use "_" to join n-gram features
    - CRFSuite uses whitespace separated feature format

- If reach edge of sentence, return partial n-gram:
    - `trigram("I like pie", 1)` → `trigram[-1]=I_like, trigram[0]=I_like_pi, trigram[1]=like_pie, ...`

```
trigram("I like pie", 1)   →   trigram[-1]=I_like,
                               trigram[0]=I_like_pi,
                               trigram[1]=like_pie,
                               ...
```

CRFSuite Evaluation

```
***** Iteration #197 *****
Active features: 514096
Performance by label (#match, #model, #ref) (precision, recall, F1):
    O: (15076, 15276, 15163) (0.9869, 0.9943, 0.9906)
    B-GPE: (131, 164, 171) (0.7988, 0.7661, 0.7821)
    B-ORG: (103, 143, 195) (0.7203, 0.5282, 0.6095)
    B-PER: (268, 333, 330) (0.8048, 0.8121, 0.8084)
    I-PER: (118, 143, 132) (0.8252, 0.8939, 0.8582)
    B-LOC: (4, 5, 16) (0.8000, 0.2500, 0.3810)
    I-LOC: (4, 5, 6) (0.8000, 0.6667, 0.7273)
    I-ORG: (128, 172, 196) (0.7442, 0.6531, 0.6957)
    B-FAC: (1, 1, 10) (1.0000, 0.1000, 0.1818)
    I-FAC: (1, 1, 12) (1.0000, 0.0833, 0.1538)
    I-GPE: (19, 23, 35) (0.8261, 0.5429, 0.6552)
Macro-average precision, recall, F1: (0.620415, 0.419369, 0.456231)
Item accuracy: 15853 / 16266 (0.9746)
Total seconds required for training: 52.877
```

Results

	Precision	**Recall**	**F1**
Development	**0.7883**	0.6809	0.7307
Test	0.7848	**0.7044**	**0.7425**

Bibliography

1. Naoaki Okazaki (2007). Crfsuite: a fast implementation of conditional random fields (crfs). *URL: http://www.chokkan.org/software/crfsuite/*
2. Mikolov, G. C. T., K. Chen, and J. Dean (2013). word2vec. *URL: https://code.google.com/p/word2vec/*
3. Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., & Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, *18*(4), 467-479.