# NLP

## Lab 2: Named Entity Recognition

Due date: 11/21

# Outline

- You have to train a Chinese NER model using sklearn-crfsuite module and evaluate its performance. Character-based Chinese NER dataset in json format will be given.

- Try to improve model performance using suggested methods and compare the results.

# Procedure

# sklearn-crfsuite

## Feature Engineering
## (Please refer to Featuring suggestions)

```
train_sents[0]
```

```
[('Melbourne', 'NP', 'B-LOC'),
 ('(', 'Fpa', 'O'),
 ('Australia', 'NP', 'B-LOC'),
 (')', 'Fpt', 'O'),
 (',', 'Fc', 'O'),
 ('25', 'Z', 'O'),
 ('may', 'NC', 'O'),
 ('(', 'Fpa', 'O'),
 ('EFE', 'NC', 'B-ORG'),
 (')', 'Fpt', 'O'),
 ('.', 'Fp', 'O')]
```

```
def word2features(sent, i):
    word = sent[i][0]
    postag = sent[i][1]

    features = {
        'bias': 1.0,
        'word.lower()': word.lower(),
        'word[-3:]': word[-3:],
        'word[-2:]': word[-2:],
        'word.isupper()': word.isupper(),
        'word.istitle()': word.istitle(),
        'word.isdigit()': word.isdigit(),
        'postag': postag,
        'postag[:2]': postag[:2],
    }
    if i > 0:
        word1 = sent[i-1][0]
        postag1 = sent[i-1][1]
        features.update({
            '-1:word.lower()': word1.lower(),
            '-1:word.istitle()': word1.istitle(),
            '-1:word.isupper()': word1.isupper(),
            '-1:postag': postag1,
            '-1:postag[:2]': postag1[:2],
        })
    else:
        features['BOS'] = True

    if i < len(sent)-1:
        word1 = sent[i+1][0]
        postag1 = sent[i+1][1]
        features.update({
            '+1:word.lower()': word1.lower(),
            '+1:word.istitle()': word1.istitle(),
            '+1:word.isupper()': word1.isupper(),
            '+1:postag': postag1,
            '+1:postag[:2]': postag1[:2],
        })
    else:
        features['EOS'] = True

    return features
```

# sklearn-crfsuite

## Assign feature as training data

```
%%time
X_train = [sent2features(s) for s in train_sents]
y_train = [sent2labels(s) for s in train_sents]

X_test = [sent2features(s) for s in test_sents]
y_test = [sent2labels(s) for s in test_sents]
```

## Training

```
%%time
crf = sklearn_crfsuite.CRF(
    algorithm='lbfgs',
    c1=0.1,
    c2=0.1,
    max_iterations=100,
    all_possible_transitions=True
)
crf.fit(X_train, y_train)
```

# sklearn-crfsuite

## Evaluation

```
y_pred = crf.predict(X_test)
metrics.flat_f1_score(y_test, y_pred,
                      average='weighted', labels=labels)
```

```
0.76980231377134023
```

# sklearn-crfsuite

## Evaluation - more detail

```
# group B and I results
sorted_labels = sorted(
    labels,
    key=lambda name: (name[1:], name[0])
)
print(metrics.flat_classification_report(
    y_test, y_pred, labels=sorted_labels, digits=3
))
```

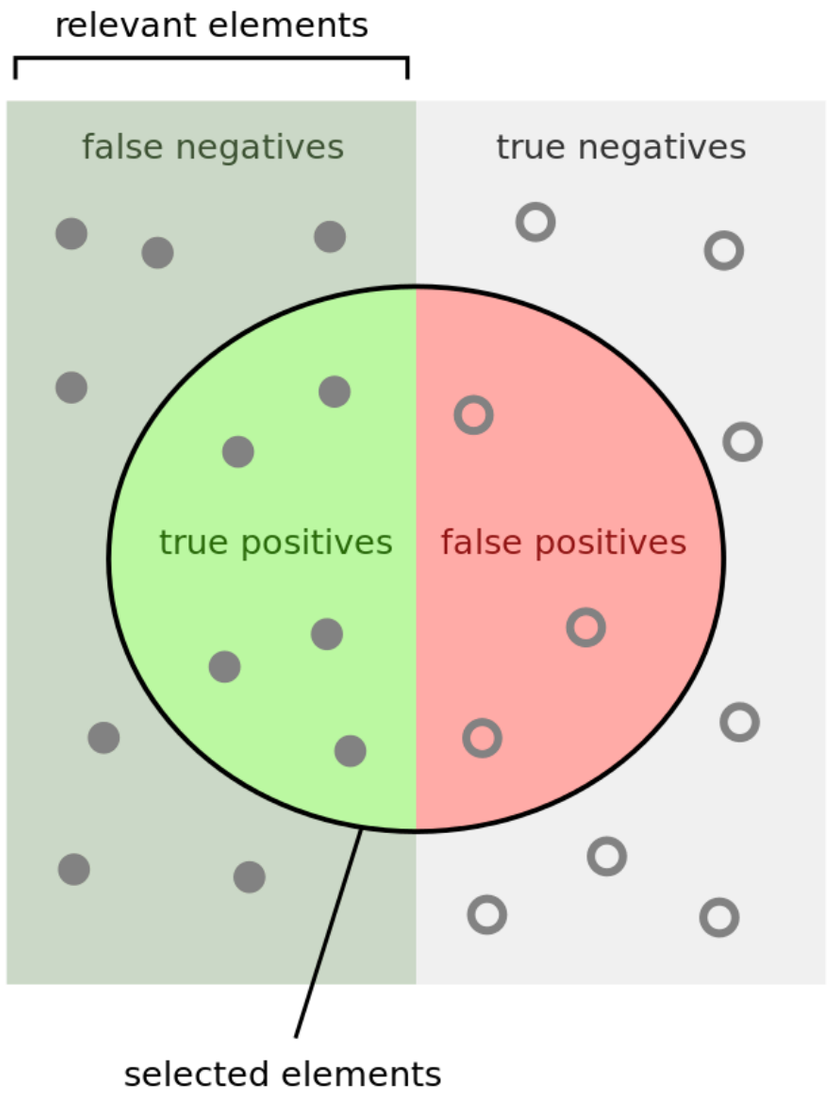
```
             precision    recall  f1-score   support

      B-LOC      0.775     0.757     0.766      1084
      I-LOC      0.601     0.631     0.616       325
     B-MISC      0.698     0.499     0.582       339
     I-MISC      0.644     0.567     0.603       557
      B-ORG      0.795     0.801     0.798      1400
      I-ORG      0.831     0.773     0.801      1104
      B-PER      0.812     0.876     0.843       735
      I-PER      0.873     0.931     0.901       634

avg / total      0.779     0.764     0.770      6178
```

# sklearn-crfsuite

## Evaluation




| Predicted Value (predicted by the test) | | Actual Value (as confirmed by experiment): positives | Actual Value (as confirmed by experiment): negatives |
|---|---|---|---|
| | positives | **TP** True Positive | **FP** False Positive |
| | negatives | **FN** False Negative | **TN** True Negative |

How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$F_1 = \left( \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

## Char-based

```
正 O
如 O
宋 B_Time
代 I_Time
詩 B_Person
人 I_Person
高 I_Person
翥 I_Person
所 O
雲 O
“ O
南 B_Location
北 I_Location
山 I_Location
頭 I_Location
多 O
墓 B_Thing
田 I_Thing
， O
清 B_Time
明 I_Time
祭 O
掃 O
各 O
紛 O
然 O
。 O
```

## Word-based

```
正 D
如 P
宋代 Nd
詩人高翥 Nb
所 Nc
雲 Na
“ FW
南北山頭 Nc
多 D
墓田 Nb
， COMMACATEGORY
清明 Nd
祭掃 VC
各 Nes
紛然 VH
。 PERIODCATEGORY
```

**Char-based: 27 chars; 14 positive, 13 negative.**

**Word-based: 16 words; 5 positive, 11 negative.**

# dataset

清明是人們祭掃先人，懷念追思的日子。
正如宋代詩人高翥所雲“南北山頭多墓田，清明祭掃
紙灰飛作白蝴蝶，淚血染成紅杜鵑”。
凡清明之時，總是屢屢哀思湧上心頭，對母親懷念的
母親姓孫名諱秋蘭，所以，我特別喜歡陳毅“幽蘭在

```
[('正', 'D', 'O'),
 ('如', 'P', 'O'),
 ('宋', 'Nd', 'B_Time')
 ('代', 'Nd', 'I_Time')
 ('詩', 'Na', 'B_Person'),
 ('人', 'Na', 'I_Person'),
 ('高', 'Nb', 'I_Person'),
 ('翥', 'Nb', 'I_Person'),
 ('所', 'Nc', 'O'),
 ('雲', 'Na', 'O'),
 ('“', 'FW', 'O'),
 ('南', 'Ncd', 'B_Location'),
 ('北', 'Ncd', 'I_Location'),
 ('山', 'Nc', 'I_Location'),
 ('頭', 'Nc', 'I_Location'),
 ('多', 'Nb', 'O'),
 ('墓', 'Nb', 'B_Thing')
 ('田', 'Nb', 'I_Thing')
 ('，', 'COMMACATEGORY', 'O'),
 ('清', 'Nd', 'B_Time'),
 ('明', 'Nd', 'I_Time'),
 ('祭', 'VC', 'O'),
 ('掃', 'VC', 'O'),
 ('各', 'Nes', 'O'),
 ('紛', 'VH', 'O'),
 ('然', 'VH', 'O'),
 ('。', 'PERIODCATEGORY', 'O')]
```

- train_char.json
- validation_char.json
- test_char.json

`[token, pos, neLabel]`

# dataset

清明是人們祭掃先人，懷念追思的日子。
正如宋代詩人高翥所雲“南北山頭多墓田，清明祭掃各紛然。
紙灰飛作白蝴蝶，淚血染成紅杜鵑”。
凡清明之時，總是屢屢哀思湧上心頭，對母親懷念的情愫越發細膩綿長。
母親姓孫名諱秋蘭，所以，我特別喜歡陳毅“幽蘭在山谷，本自無人識。

- train_
- valida
- test_w


```
[["清明 _ Time", "是 SHI 0", "人們 _ Person", "祭掃 VC 0",
 "先人 _ Person", "， COMMACATEGORY 0", "懷念 VJ 0", "追思
 VK 0", "的 DE 0", "日子 _ Time", "。 PERIODCATEGORY 0"],
["正 D 0", "如 P 0", "宋代 _ Time", "詩人高翥 _ Person", "
所 Nc 0", "雲 Na 0", "“ FW 0", "南北山頭 _ Location", "多
VH 0", "墓田 _ Thing", "， COMMACATEGORY 0", "清明 _ Time"
, "祭掃 VC 0", "各 Nes 0", "紛然 VH 0", "。 PERIODCATEGORY
 0"], ["紙灰飛 _ Thing", "作 VC 0", "白蝴蝶 _ Thing", "，
```


**[token, pos, neLabel]**

# Featuring suggestions

1. Refer to sample code method: part-of-speech, upper / lower / titlecase, suffix, word length... (use your imagination). (Required)
2. Change from character-based to word-based. (Bonus)
3. Word2Vec pretrained model. (Required)
4. Word2Vec self-trained model. (Required)

# Word2Vec

- Pre-trained model:

  - Trained from 中央社CNA.
    File: cna.word2vec.bin
    Dimension: 300.

- Self-trained model:

  - We already trained the Word2Vec model for you.
    Training from the Chinese NER data given.
    File: ch.300.bin
    Dimension: 300

# Word2Vec

- Gensim documentation: https://radimrehurek.com/gensim/models/keyedvectors.html

```
[13]: import gensim
      from gensim.models import Word2Vec
      from gensim.models import KeyedVectors

[16]: word2vec = KeyedVectors.load_word2vec_format("/PATH/TO/ch.300.bin", binary=True)

[17]: word2vec.wv['清明']

      /usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: Call to de
      precated `wv` (Attribute will be removed in 4.0.0, use self instead).
        """Entry point for launching an IPython kernel.

[17]: array([-0.05964928,  0.01020973,  0.00801982,  0.04241873,  0.01073279,
             -0.01942019, -0.05009415, -0.07332227, -0.09036233,  0.26621333,
             -0.12485693, -0.15073214, -0.07478274,  0.2253082 , -0.1424772 ,
             -0.33393162, -0.02561557,  0.03427131, -0.1750736 ,  0.17938018,
             -0.1001088 ,  0.00432608, -0.08760878, -0.05636474, -0.0145522 ,
              0.06306485, -0.07776728,  0.01629716,  0.00956435,  0.0684054 ,
              0.189433  ,  0.26781738, -0.01246896,  0.02744563, -0.03833575,
             -0.0319246 ,  0.04408965, -0.06920365, -0.40958452,  0.04468812,
             -0.18849126,  0.2800773 ,  0.01218978,  0.42352524,  0.14850967,
              0.04358063,  0.0421218 , -0.07971724, -0.1287401 , -0.12652162,
             -0.04743549, -0.12770219, -0.0420709 ,  0.09501084, -0.10525673,
             -0.01056469,  0.03603829,  0.06542812,  0.02837644, -0.14888528,
              0.05441643, -0.17309391, -0.06392956, -0.20448618,  0.0338928 ,
              0.10788038, -0.01556227,  0.0930739 , -0.02076078, -0.0168945 ,
             -0.02359504, -0.13476035, -0.06226062,  0.06476686,  0.00617765,
             -0.03893873,  0.03196216, -0.17000991, -0.28195596, -0.06585351,
```

# Grading

- Code; Featuring method 50%
- Report: result comparison 40%
- Highest F1 score 10%