In [ ]:

```
from itertools import chain

import matplotlib.pyplot as plt
import nltk
import numpy as np
import pandas as pd
import scipy.stats
import sklearn
import sklearn_crfsuite
from sklearn import metrics as mt
from sklearn.metrics import make_scorer
from sklearn.model_selection import RandomizedSearchCV, cross_val_score
from sklearn_crfsuite import metrics, scorers
from sklearn_crfsuite.utils import flatten

TRAIN_FILE_NAME = "04-train.txt"
TEST_FILE_NAME = "04-test.txt"
```

A simple sentence NER example:

[**ORG** U.N.] official [**PER** Ekeus] heads for [**LOC** Baghdad]

We will concentrate on four types of named entities:

- persons (**PER**),
- locations (**LOC**)
- organizations (**ORG**)
- Others (**O**)

In [ ]:

```
def _generate_examples(filepath):
    with open(filepath, encoding="utf-8") as f:
        sent = []
        for line in f:
            if line.startswith("-DOCSTART-") or line == "" or line == "\n":
                if sent:
                    yield sent
                    sent = []
            else:
```

```
                splits = line.split(" ")
                token = splits[0]
                pos_tag = splits[1]
                ner_tag = splits[3].rstrip()
                if "MISC" in ner_tag:
                    ner_tag = "O"

                sent.append((token, pos_tag, ner_tag))
```

In [ ]:

```
%%time
# hint use the above defined function
train_sents = list(_generate_examples(TRAIN_FILE_NAME))
test_sents = list(_generate_examples(TEST_FILE_NAME))
```

```
CPU times: user 163 ms, sys: 16.8 ms, total: 179 ms
Wall time: 178 ms
```

In [ ]:

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

    features = {
        "bias": 1.0,
        "word.lower()": word.lower(),
        "postag": postag,
    }

    if i > 0:
        word1 = sent[i - 1][0]
        postag1 = sent[i - 1][1]
        features.update(
            {
                "-1:word.lower()": word1.lower(),
                "-1:postag": postag1,
            }
        )
    else:
        features["BOS"] = True
    return features
```

In [ ]:

```
test_sents[2]
```

Out[ ]:
```
[('United', 'NNP', 'B-LOC'),
 ('Arab', 'NNP', 'I-LOC'),
 ('Emirates', 'NNPS', 'I-LOC'),
 ('1996-12-06', 'CD', 'O')]
```

In [ ]:
```
word2features(test_sents[2], 0)
```

Out[ ]:
```
{'bias': 1.0, 'word.lower()': 'united', 'postag': 'NNP', 'BOS': True}
```

In [ ]:
```
def sent2features(sent):
    return [word2features(sent, i) for i in range(len(sent))]


def sent2labels(sent):
    return [label for token, postag, label in sent]


def sent2tokens(sent):
    return [token for token, postag, label in sent]
```

In [ ]:
```
%%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]
```

```
CPU times: user 303 ms, sys: 19.3 ms, total: 322 ms
Wall time: 320 ms
```

In [ ]:
```
%%time
#search for sklearn_crfsuite.CRF,
# use the lbfgs algorithm,
# c parameters should be 0.1 and max iterations 100,
# all possible transactions true
try:
    crf = sklearn_crfsuite.CRF(algorithm="lbfgs", c1=0.1, c2=0.1, max_iterations=100, all_possible_transitions=True,)
```

```
    # fit the model
    crf.fit(X_train, y_train)
except AttributeError as e:
    print("Error", e)
```

```
CPU times: user 10.4 s, sys: 33.4 ms, total: 10.4 s
Wall time: 10.4 s
```

In [ ]:

```
# save a list of all labels in your model, hint crfs have a classes attribute
labels = list(crf.classes_)
labels
```

Out[ ]:

```
['B-ORG', 'O', 'B-PER', 'I-PER', 'B-LOC', 'I-ORG', 'I-LOC']
```

In [ ]:

```
# remove the label 'O' from your list
try:
    labels.remove("O")
except ValueError:
    pass
labels
```

Out[ ]:

```
['B-ORG', 'B-PER', 'I-PER', 'B-LOC', 'I-ORG', 'I-LOC']
```

In [ ]:

```
# perfrom a prediction on your test set
y_pred = crf.predict(X_test)

metrics.flat_f1_score(
    y_test,
    y_pred,
    average="weighted",
    labels=labels,
)
```

Out[ ]:

```
0.7757476721426669
```

In [ ]:

```
# group B and I results, use the sorted function on the list labels with a lambda function as the key
sorted_labels =sorted(labels,key=lambda l1: (l1[1:], l1[0]))
```

In [ ]:

```
# Display classification report
print(
    mt.classification_report(
        y_true=flatten(y_test),
        y_pred=flatten(y_pred),
        labels=sorted_labels,
        digits=3,
    )
)
```

```
              precision    recall  f1-score   support

       B-LOC      0.849     0.825     0.837      1667
       I-LOC      0.767     0.716     0.740       257
       B-ORG      0.735     0.637     0.682      1660
       I-ORG      0.616     0.721     0.664       834
       B-PER      0.837     0.764     0.799      1615
       I-PER      0.832     0.931     0.878      1156

   micro avg      0.785     0.769     0.777      7189
   macro avg      0.772     0.766     0.767      7189
weighted avg      0.787     0.769     0.776      7189
```

In [ ]:

```
# what is the number of transition features in our model, crfs have an attribute called transition_features_
len(crf.transition_features_)
```

Out[ ]:

```
49
```

In [ ]:

```
from collections import Counter


def print_transitions(trans_features):
    for (label_from, label_to), weight in trans_features:
        print("%-6s ->  %-7s %0.6f" % (label_from, label_to, weight))


print("Top likely transitions:")
print_transitions(Counter(crf.transition_features_).most_common(20))

# top 20 unlikely transitions
print("\nTop unlikely transitions:")
```

```
(
    pd.DataFrame(crf.transition_features_, index=["value"])
    .transpose()
    .reset_index()
    .rename(
        columns={
            "level_0": "from",
            "level_1": "to",
        },
    )
    .sort_values(by="value")
    .reset_index(drop=True)
    .head(20)
)
```

```
Top likely transitions:
B-PER  ->  I-PER   6.591492
B-ORG  ->  I-ORG   6.306534
I-ORG  ->  I-ORG   5.540077
B-LOC  ->  I-LOC   4.839887
I-LOC  ->  I-LOC   3.758774
I-PER  ->  I-PER   3.394919
O      ->  B-PER   1.960743
O      ->  O       1.369676
B-ORG  ->  O       0.950664
O      ->  B-LOC   0.919982
B-LOC  ->  O       0.612921
B-PER  ->  O       0.557646
O      ->  B-ORG   0.515605
I-PER  ->  O       0.393510
I-ORG  ->  O       0.328486
I-LOC  ->  O       -0.305074
B-ORG  ->  B-ORG   -0.984217
B-LOC  ->  B-LOC   -0.990422
I-LOC  ->  B-LOC   -1.291094
B-PER  ->  B-LOC   -1.315197

Top unlikely transitions:
```

Out[ ]:

	from	to	value
0	O	I-ORG	-6.175097
1	O	I-LOC	-5.995912
2	O	I-PER	-4.319554
3	B-LOC	I-ORG	-3.856252
4	B-PER	B-PER	-3.500576
5	I-PER	B-PER	-3.186294
6	I-PER	B-ORG	-3.144592
7	I-ORG	B-ORG	-3.030456
8	B-PER	B-ORG	-2.798284
9	I-LOC	I-ORG	-2.759703
10	I-LOC	B-PER	-2.756531
11	B-LOC	B-PER	-2.738535
12	I-PER	B-LOC	-2.709911
13	I-LOC	B-ORG	-2.658712
14	B-ORG	B-LOC	-2.584979
15	I-ORG	I-LOC	-2.206704
16	I-ORG	B-LOC	-2.113390
17	B-ORG	I-LOC	-2.046612
18	I-PER	I-LOC	-2.040264
19	I-PER	I-ORG	-1.991960

In [ ]:

```
# number of transition features in our model
len(crf.state_features_)
```

Out[ ]: 16044

In [ ]:

```
# create dataframe to easily sort linked values
```

```
df_trans = (
    pd.DataFrame(crf.state_features_, index=["value"])
    .transpose()
    .reset_index()
    .rename(
        columns={
            "level_0": "attr_name",
            "level_1": "label",
        },
    )
)
df_trans = df_trans[["value", "label", "attr_name"]]
```

In [ ]:

```
def print_state_features(state_features):
    for (attr, label), weight in state_features:
        print("%0.6f %-8s %s" % (weight, label, attr))


# top 30 positive
print("Top positive:")
display(
    df_trans.sort_values(
        by="value",
        ascending=False,
        ignore_index=True,
    ).head(30)
)


# top 30 negative
print("\nTop negative:")
display(
    df_trans.sort_values(
        by="value",
        ignore_index=True,
    ).head(30)
)
```

```
Top positive:
```

	value	label	attr_name
0	8.307293	I-LOC	word.lower():oval
1	8.088441	B-LOC	word.lower():m3
2	7.751193	B-ORG	word.lower():footscray
3	7.001409	B-ORG	word.lower():osce
4	6.964246	B-PER	word.lower():lebed
5	6.609227	B-LOC	word.lower():amsterdam
6	6.556081	B-LOC	word.lower():bonn
7	6.543649	B-LOC	word.lower():beijing
8	6.516252	B-LOC	word.lower():mideast
9	6.514129	B-ORG	word.lower():adelaide
10	6.341000	B-LOC	word.lower():balkans
11	6.296972	B-LOC	word.lower():med
12	6.293931	B-LOC	word.lower():stansted
13	6.251496	O	word.lower():to
14	6.164668	O	word.lower():division
15	6.109046	B-LOC	word.lower():vatican
16	6.065353	B-LOC	word.lower():johannesburg
17	6.051745	B-PER	word.lower():stenning
18	6.033868	B-LOC	word.lower():england
19	6.013143	B-PER	word.lower():clinton
20	5.989218	B-PER	word.lower():chang
21	5.986145	B-LOC	word.lower():pakistan
22	5.979900	B-LOC	word.lower():mt
23	5.896624	B-PER	word.lower():fogarty
24	5.840211	B-LOC	word.lower():moscow
25	5.801627	B-ORG	word.lower():u.n.

	value	label	attr_name
26	5.759773	B-LOC	word.lower():seoul
27	5.753325	B-LOC	word.lower():iraq
28	5.746310	I-ORG	word.lower():newsroom
29	5.683387	B-LOC	word.lower():beirut

```
Top negative:
```

	value	label	attr_name
0	-4.085484	O	-1:word.lower():lloyd
1	-3.924442	O	postag:NNP
2	-3.343803	O	-1:word.lower():beat
3	-3.240531	O	postag:NNPS
4	-3.216858	O	-1:word.lower():st
5	-2.649809	O	-1:word.lower():queen
6	-2.587832	O	-1:word.lower():moody
7	-2.562613	O	word.lower():leeds
8	-2.527634	O	-1:word.lower():buducnost
9	-2.493048	I-PER	bias
10	-2.397402	O	postag:TO
11	-2.321810	O	word.lower():nice
12	-2.296212	O	-1:word.lower():awami
13	-2.283222	O	-1:word.lower():cdu
14	-2.247559	O	word.lower():ours
15	-2.190894	O	-1:word.lower():saint
16	-2.161106	O	-1:word.lower():past
17	-2.116170	B-PER	-1:word.lower():/
18	-2.092937	O	-1:word.lower():n
19	-2.088915	B-PER	-1:postag:PRP$
20	-2.078149	O	-1:word.lower():diario
21	-2.076839	O	-1:word.lower():p
22	-2.071485	O	word.lower():31
23	-2.062987	O	-1:word.lower():breaking
24	-2.034884	O	-1:word.lower():arkansas
25	-1.990570	B-ORG	word.lower():african

	value	label	attr_name
26	-1.976188	I-PER	postag:VBD
27	-1.968579	O	-1:word.lower():later
28	-1.934276	O	-1:word.lower():cooperation
29	-1.924981	O	-1:word.lower():colleague