[1]:
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import GroupShuffleSplit
from hmmlearn import hmm
from sklearn.metrics import confusion_matrix, classification_report,␣
↪accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
```

0.0.3 Loading Data

[35]:
```python
data = pd.read_csv("ner_data.csv", encoding='latin1')
data.head()
```

[35]:

	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	NaN	of	IN	O
2	NaN	demonstrators	NNS	O
3	NaN	have	VBP	O
4	NaN	marched	VBN	O

0.0.4 Data Cleaning

[36]:
```python
data = data.fillna(method="ffill")
data = data.rename(columns={'Sentence #': 'sentence'})
data.head(5)
```

[36]:

	sentence	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	Sentence: 1	of	IN	O
2	Sentence: 1	demonstrators	NNS	O
3	Sentence: 1	have	VBP	O
4	Sentence: 1	marched	VBN	O

[38]:
```python
' '.join(data[data['sentence'] == 'Sentence: 1'].Word.tolist())
```

[38]: 'Thousands of demonstrators have marched through London to protest the war in Iraq and demand the withdrawal of British troops from that country .'

[39]:
```
tags = list(set(data.Tag.values))
words = list(set(data.Word.values))
```

[41]:
```
print(f"Total Tags: {len(tags)}")
print()
print(tags)
```

```
Total Tags: 17

['O', 'I-geo', 'I-per', 'I-eve', 'B-gpe', 'B-art', 'B-per', 'I-gpe', 'I-nat',
'B-eve', 'I-org', 'I-tim', 'B-geo', 'I-art', 'B-org', 'B-nat', 'B-tim']
```

0.0.5 Data Preparation

We cannot split data normally with `train_test_split` because doing that makes some parts of a sentence in the training set while some others in the testing set. Instead, we use `GroupShuffleSplit`.

[42]:
```
y = data.Tag
X = data.drop('Tag', axis=1)
```

[43]:
```
gs = GroupShuffleSplit(n_splits=2, test_size=.33, random_state=42)
train_ix, test_ix = next(gs.split(X, y, groups=data['sentence']))
```

[44]:
```
data_train = data.loc[train_ix]
data_test = data.loc[test_ix]
```

[45]:
```
data_train
```

[45]:

	sentence	Word	POS	Tag
24	Sentence: 2	Families	NNS	O
25	Sentence: 2	of	IN	O
26	Sentence: 2	soldiers	NNS	O
27	Sentence: 2	killed	VBN	O
28	Sentence: 2	in	IN	O
…	…	…	…	..
1048570	Sentence: 47959	they	PRP	O
1048571	Sentence: 47959	responded	VBD	O
1048572	Sentence: 47959	to	TO	O
1048573	Sentence: 47959	the	DT	O
1048574	Sentence: 47959	attack	NN	O

[702936 rows x 4 columns]

[46]:
```
tags = list(set(data_train.Tag.values))
words = list(set(data_train.Word.values))
```

[47]:
```
dfupdate = data_train.sample(frac=.15, replace=False, random_state=42)
dfupdate.Word = 'UNKNOWN'
data_train.update(dfupdate)
```

[48]:
```
words = list(set(data_train.Word.values))
word2id = {w: i for i, w in enumerate(words)}
tag2id = {t: i for i, t in enumerate(tags)}
id2tag = {i: t for i, t in enumerate(tags)}
```

0.0.6 Model Parameter Estimation

Hidden Markov Models can be trained by using the Baum-Welch algorithm. - startprob_ - transmat_ - emissionprob_

[49]:
```
count_tags = dict(data_train.Tag.value_counts())
count_tags_to_words = data_train.groupby(['Tag']).apply(lambda grp: grp.
 ↪groupby('Word')['Tag'].count().to_dict()).to_dict()
count_init_tags = dict(data_train.groupby('sentence').first().Tag.
 ↪value_counts())
```

[50]:
```
count_tags_to_next_tags = np.zeros((len(tags), len(tags)), dtype=int)
sentences = list(data_train.sentence)
pos = list(data_train.Tag)
for i in range(len(sentences)) :
    if (i > 0) and (sentences[i] == sentences[i - 1]):
        prevtagid = tag2id[pos[i - 1]]
        nexttagid = tag2id[pos[i]]
        count_tags_to_next_tags[prevtagid][nexttagid] += 1
```

Calculating Probablites

[51]:
```
mystartprob = np.zeros((len(tags),))
mytransmat = np.zeros((len(tags), len(tags)))
myemissionprob = np.zeros((len(tags), len(words)))
num_sentences = sum(count_init_tags.values())
sum_tags_to_next_tags = np.sum(count_tags_to_next_tags, axis=1)
for tag, tagid in tag2id.items():
    floatCountTag = float(count_tags.get(tag, 0))
    mystartprob[tagid] = count_init_tags.get(tag, 0) / num_sentences
    for word, wordid in word2id.items():
        myemissionprob[tagid][wordid]= count_tags_to_words.get(tag, {}).
 ↪get(word, 0) / floatCountTag
    for tag2, tagid2 in tag2id.items():
```

```
        mytransmat[tagid][tagid2]= count_tags_to_next_tags[tagid][tagid2] /␣
↪sum_tags_to_next_tags[tagid]
```

[52]:
```
mystartprob
```

[52]:
```
array([7.16419768e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       6.27100710e-02, 3.42337856e-04, 8.25967882e-02, 0.00000000e+00,
       0.00000000e+00, 2.48972986e-04, 0.00000000e+00, 0.00000000e+00,
       6.89966389e-02, 0.00000000e+00, 5.80418275e-02, 2.17851363e-04,
       1.04257438e-02])
```

[53]:
```
mytransmat
```

[53]:
```
array([[8.89746990e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.48244413e-02, 4.33193256e-04, 1.28715209e-02, 0.00000000e+00,
        0.00000000e+00, 3.00039591e-04, 0.00000000e+00, 0.00000000e+00,
        4.03011758e-02, 0.00000000e+00, 1.93978259e-02, 2.20147392e-04,
        2.19046655e-02],
       [8.70028124e-01, 1.05062274e-01, 0.00000000e+00, 0.00000000e+00,
        4.01767778e-04, 2.00883889e-04, 3.01325834e-03, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 1.80795500e-03, 0.00000000e+00,
        1.94857372e-02],
       [7.18293954e-01, 0.00000000e+00, 2.71195274e-01, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        8.68658791e-05, 0.00000000e+00, 6.94927033e-04, 0.00000000e+00,
        9.72897846e-03],
       [6.95121951e-01, 0.00000000e+00, 0.00000000e+00, 2.92682927e-01,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.21951220e-02],
       [8.69222097e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 9.39496430e-05, 8.25817362e-02, 1.28711011e-02,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        7.70387073e-03, 0.00000000e+00, 2.57422022e-02, 0.00000000e+00,
        1.78504322e-03],
       [5.54716981e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 3.77358491e-03, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 4.11320755e-01, 1.13207547e-02, 0.00000000e+00,
        1.88679245e-02],
       [2.18449599e-01, 0.00000000e+00, 7.40717876e-01, 0.00000000e+00,
        3.35126554e-03, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.47279302e-02, 0.00000000e+00, 2.01957845e-02, 8.81911985e-05,
```

```
 2.46935356e-03],
[8.74125874e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 7.69230769e-02, 4.19580420e-02,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 6.99300699e-03, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00],
[9.00000000e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 3.33333333e-02, 0.00000000e+00,
 6.66666667e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00],
[4.06091371e-01, 0.00000000e+00, 0.00000000e+00, 5.88832487e-01,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 5.07614213e-03],
[5.27861848e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 2.49243368e-03, 0.00000000e+00, 1.98504540e-02, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 4.32971337e-01, 0.00000000e+00,
 1.42424782e-03, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 1.53996795e-02],
[6.90530131e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 1.58586316e-03, 0.00000000e+00, 2.49207068e-03, 0.00000000e+00,
 0.00000000e+00, 1.58586316e-03, 0.00000000e+00, 2.94517444e-01,
 7.92931581e-03, 0.00000000e+00, 1.35931128e-03, 0.00000000e+00,
 0.00000000e+00],
[7.94497229e-01, 1.76484561e-01, 0.00000000e+00, 0.00000000e+00,
 4.15676960e-03, 0.00000000e+00, 2.85035629e-03, 0.00000000e+00,
 0.00000000e+00, 3.95882819e-05, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 1.38558987e-03, 0.00000000e+00,
 2.05859066e-02],
[6.06936416e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 1.15606936e-02, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 3.69942197e-01, 0.00000000e+00, 0.00000000e+00,
 1.15606936e-02],
[4.97683109e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 5.68011958e-03, 3.73692078e-04, 1.49476831e-02, 0.00000000e+00,
 0.00000000e+00, 1.49476831e-04, 4.76457399e-01, 0.00000000e+00,
 8.96860987e-04, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 3.81165919e-03],
[7.80303030e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 2.12121212e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 7.57575758e-03, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
 0.00000000e+00],
[7.61915205e-01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
```

```
          1.68128655e-03, 2.19298246e-04, 1.60818713e-03, 0.00000000e+00,
          0.00000000e+00, 7.30994152e-04, 0.00000000e+00, 2.27923977e-01,
          4.02046784e-03, 0.00000000e+00, 1.90058480e-03, 0.00000000e+00,
          0.00000000e+00]])
```

[54]:
```
myemissionprob
```

[54]:
```
array([[0.00000000e+00, 0.00000000e+00, 1.67972090e-06, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 8.67980210e-05, 0.00000000e+00, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       …,
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [4.38500329e-04, 0.00000000e+00, 0.00000000e+00, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

0.0.7 HMM

[55]:
```
model = hmm.MultinomialHMM(n_components=len(tags), algorithm='viterbi',␣
 ↪random_state=42)
model.startprob_ = mystartprob
model.transmat_ = mytransmat
model.emissionprob_ = myemissionprob
```

[56]:
```
data_test.loc[~data_test['Word'].isin(words), 'Word'] = 'UNKNOWN'
```

[57]:
```
word_test = list(data_test.Word)
samples = []
for i, val in enumerate(word_test):
    samples.append([word2id[val]])
```

[58]:
```
len(samples)
```

[58]: 345639

[59]:
```
lengths = []
count = 0
sentences = list(data_test.sentence)
for i in range(len(sentences)) :
    if (i > 0) and (sentences[i] == sentences[i - 1]):
        count += 1
    elif i > 0:
```

```
        lengths.append(count)
        count = 1
    else:
        count = 1
```

[60]:
```
len(lengths)
```

[60]: 15826

0.0.8 HMM - Prediction

[61]:
```
ner_predict = model.predict(samples, lengths)
```

[62]:
```
ner_predict
```

[62]: array([0, 0, 0, …, 12, 0, 0], dtype=int32)

0.0.9 Testing

[64]:
```
def reportTest(y_pred, y_test):
    print("The accuracy is {}".format(accuracy_score(y_test, y_pred)))
    print("The precision is {}".format(precision_score(y_test, y_pred,␣
 ↪average='weighted')))
    print("The recall is {}".format(recall_score(y_test, y_pred,␣
 ↪average='weighted')))
    print("The F1-Score is {}".format(f1_score(y_test, y_pred,␣
 ↪average='weighted')))

min_length = min(len(pos_predict), len(pos_test))

reportTest(pos_predict[:min_length], pos_test[:min_length])
```

```
The accuracy is 0.9560956555705048
The precision is 0.9544313510538452
The recall is 0.9560956555705048
The F1-Score is 0.9547968981401819
```