Sequence Learning

End Course Test 1

edureka!

DESCRIPTION

Predict named entity from given data and showcase how entities like persons, locations, organizations, and other miscellaneous entity names of that do not belong to the previous three groups can be predicted from the tagged dataset.

Problem Statement:

Using CoNLL 2003 data, predict tagged NER using CRF Algorithm. Also, tune the algorithm and explore the learnings that have been done by the CRF Model.

Dataset:

The CoNLL-2003 shared task data files contain four columns separated by a single space. Each word has been put on a separate line, and there is an empty line after each sentence. The first item on each line is a word, the second a part-of-speech (POS) tag, the third a syntactic chunk tag, and the fourth the named entity tag. The chunk tags and the named entity tags have the format I-TYPE, which means that the word is inside a phrase of type TYPE. Only if two phrases of the same type immediately follow each other, the first word of the second phrase will have tag B-TYPE to show that it starts a new phrase. A word with tag O is not part of a phrase. Here is an example:

```
EU NNP B-NP B-ORG
rejects VBZ B-VP O
German JJ B-NP B-MISC
call NN I-NP O
to TO B-VP O
boycott VB I-VP O
British JJ B-NP B-MISC
lamb NN I-NP O
. . O O
```

Source: https://www.clips.uantwerpen.be/conll2003/ner/

Different Classes:

1. persons, (`PER`)
2. locations, (`LOC`)
3. Organizations (`ORG`) ,
4. names of miscellaneous entities that do not belong to the previous three groups (`MISC`)

Set 1:

Question 1.1:

Read the tagged ConLL 2003 Data from this link:

https://raw.githubusercontent.com/davidsbatista/NER-datasets/master/CONLL2003/train.txt

Read the test data from this link:

https://raw.githubusercontent.com/davidsbatista/NER-datasets/master/CONLL2003/test.txt

Read each line of the text and append in list

```
[b'-DOCSTART- -X- -X- O\n',
 b'\n',
 b'EU NNP B-NP B-ORG\n',
 b'rejects VBZ B-VP O\n',
 b'German JJ B-NP B-MISC\n',
 b'call NN I-NP O\n',
 b'to TO B-VP O\n',
 b'boycott VB I-VP O\n',
 b'British JJ B-NP B-MISC\n',
 b'lamb NN I-NP O\n',
 b'. . O O\n',
 b'\n',
 b'Peter NNP B-NP B-PER\n',
 b'Blackburn NNP I-NP I-PER\n',
 b'\n',
 b'BRUSSELS NNP B-NP B-LOC\n',
 b'1996-08-22 CD I-NP O\n',
 b'\n',
 b'The DT B-NP O\n',
 b'European NNP I-NP B-ORG\n']
```

and write a function to convert the data in the following format:

```
[[(b'EU', b'NNP', b'B-NP', b'B-ORG'),
  (b'rejects', b'VBZ', b'B-VP', b'O'),
  (b'German', b'JJ', b'B-NP', b'B-MISC'),
  (b'call', b'NN', b'I-NP', b'O'),
  (b'to', b'TO', b'B-VP', b'O'),
  (b'boycott', b'VB', b'I-VP', b'O'),
  (b'British', b'JJ', b'B-NP', b'B-MISC'),
  (b'lamb', b'NN', b'I-NP', b'O'),
  (b'.', b'.', b'O', b'O')]]
```

Hint: Each sentence is separated by “\n”. Write a for loop that splits the list with “\n” and then creates an internal sub-list of each sentence. So the final list will be a list of all sentences. [[sentence1], [sentence2], [sentence3]]. If you ended up creating multiple sublists like this:

```
[[[(b'EU', b'NNP', b'B-NP', b'B-ORG'),
   (b'rejects', b'VBZ', b'B-VP', b'O'),
   (b'German', b'JJ', b'B-NP', b'B-MISC'),
   (b'call', b'NN', b'I-NP', b'O'),
   (b'to', b'TO', b'B-VP', b'O'),
   (b'boycott', b'VB', b'I-VP', b'O'),
   (b'British', b'JJ', b'B-NP', b'B-MISC'),
   (b'lamb', b'NN', b'I-NP', b'O'),
   (b'.', b'.', b'O', b'O')]],
 [[(b'Peter', b'NNP', b'B-NP', b'B-PER'),
   (b'Blackburn', b'NNP', b'I-NP', b'I-PER')]]]
```

You can run this:

```
final_data = []

for i in range(len(Anaotated_data)):

  final_data.append(Anaotated_data[i][0])
```

Question 1.2:

Define a CRF model with following parameters:

```
algorithm='lbfgs',

    c1=0.1,

    c2=0.1,

    max_iterations=100,
```

And fit your model. Note the time taken for this step. Also, calculate the overall F1 score.

Hint:

```
              precision    recall  f1-score   support

       B-LOC      0.882     0.863     0.873      1975
       I-LOC      0.841     0.678     0.750       397
      B-MISC      0.915     0.797     0.852      1062
      I-MISC      0.833     0.593     0.693       344
       B-ORG      0.788     0.751     0.769      1617
       I-ORG      0.786     0.791     0.788      1086
       B-PER      0.827     0.887     0.856      1639
       I-PER      0.867     0.947     0.905      1029

   micro avg      0.842     0.822     0.832      9149
   macro avg      0.842     0.788     0.811      9149
weighted avg      0.842     0.822     0.830      9149
```

```
(metrics.flat_classification_report(

    y_test, y_pred, labels=sorted_labels, digits=3

))
```

You might also want to remove “O” category

```
labels = list(crf.classes_)

labels.remove('O')
```

Question 1.3:

Now that you have done a single model - try using Gridsearch to get the best parameters for our data. Use this param grid:

```
params_space = {

    'c1': scipy.stats.expon(scale=0.5),

    'c2': scipy.stats.expon(scale=0.05),

}
```

And print the best parameter and score.

Hint:

```
rs = GridSearchCV(crf, params_space,

                        cv=3,

                        verbose=1,

                        n_jobs=-1,

                        n_iter=50,

                        scoring=f1_scorer


print('best params:', rs.best_params_)

print('best CV score:', rs.best_score_)
```

Question 1.4:

For the best model that you have picked with Grid/Random Search, try getting the most important transition (top 5) features by using the `transition_features_` method of the model class. Also, get the topmost (5) state feature using `state_features_` method that tells you the most important words for specific NER.

Hint:

```
Top likely transitions:
B-ORG  -> I-ORG   6.266228
B-LOC  -> I-LOC   5.371496
I-ORG  -> I-ORG   5.283960
B-PER  -> I-PER   5.220710
I-LOC  -> I-LOC   5.009149
B-MISC -> I-MISC  5.004955
I-MISC -> I-MISC  4.782168
O      -> O       3.150406
I-PER  -> I-PER   2.876512
```

```
Top positive:
6.225423 I-LOC    -1:word.lower():wisc
5.477376 B-LOC    +1:word.lower():1996-08-26
5.463930 B-ORG    -1:word.lower():v
5.383775 B-PER    word.lower():clinton
5.306809 B-LOC    +1:word.lower():1996-08-27
5.182405 O        word[-3:]:day
5.070092 B-LOC    +1:word.lower():1996-08-25
5.022284 B-ORG    word.lower():sungard
5.021352 B-LOC    word.lower():hungary
```

Set 2:

Question 2.1:

Read the tagged ConLL 2003 Data from this link:

https://raw.githubusercontent.com/davidsbatista/NER-datasets/master/CONLL2003/train.txt

Read the test data from this link:

https://raw.githubusercontent.com/davidsbatista/NER-datasets/master/CONLL2003/test.txt

Read each line of the text and append in list

```
[b'-DOCSTART- -X- -X- O\n',
 b'\n',
 b'EU NNP B-NP B-ORG\n',
 b'rejects VBZ B-VP O\n',
 b'German JJ B-NP B-MISC\n',
 b'call NN I-NP O\n',
 b'to TO B-VP O\n',
 b'boycott VB I-VP O\n',
 b'British JJ B-NP B-MISC\n',
 b'lamb NN I-NP O\n',
 b'. . O O\n',
 b'\n',
 b'Peter NNP B-NP B-PER\n',
 b'Blackburn NNP I-NP I-PER\n',
 b'\n',
 b'BRUSSELS NNP B-NP B-LOC\n',
 b'1996-08-22 CD I-NP O\n',
 b'\n',
 b'The DT B-NP O\n',
 b'European NNP I-NP B-ORG\n']
```

and write a function to convert the data in the following format:

```
[[(b'EU', b'NNP', b'B-NP', b'B-ORG'),
  (b'rejects', b'VBZ', b'B-VP', b'O'),
  (b'German', b'JJ', b'B-NP', b'B-MISC'),
  (b'call', b'NN', b'I-NP', b'O'),
  (b'to', b'TO', b'B-VP', b'O'),
  (b'boycott', b'VB', b'I-VP', b'O'),
  (b'British', b'JJ', b'B-NP', b'B-MISC'),
  (b'lamb', b'NN', b'I-NP', b'O'),
  (b'.', b'.', b'O', b'O')]]
```

Hint: Each sentence is separated by "\n". Write a for loop that splits the list with "\n" and then creates an internal sub-list of each sentence. So the final list will be a list of all sentences. [[sentence1], [sentence2], [sentence3]] .If you ended up creating multiple sublists like this:

```
[[[(b'EU', b'NNP', b'B-NP', b'B-ORG'),
   (b'rejects', b'VBZ', b'B-VP', b'O'),
   (b'German', b'JJ', b'B-NP', b'B-MISC'),
   (b'call', b'NN', b'I-NP', b'O'),
   (b'to', b'TO', b'B-VP', b'O'),
   (b'boycott', b'VB', b'I-VP', b'O'),
   (b'British', b'JJ', b'B-NP', b'B-MISC'),
   (b'lamb', b'NN', b'I-NP', b'O'),
   (b'.', b'.', b'O', b'O')]],
 [[(b'Peter', b'NNP', b'B-NP', b'B-PER'),
   (b'Blackburn', b'NNP', b'I-NP', b'I-PER')]]]
```

You can run this:

```
final_data = []

for i in range(len(Anaotated_data)):

  final_data.append(Anaotated_data[i][0])
```

Question 2.2:

Define a CRF model with following parameters:

```
algorithm='lbfgs',

    c1=0.5,

    c2=0.5,

    max_iterations=400,
```

And fit your model. Note the time taken for this step. Also, calculate the overall F1 score.

Hint:

```
              precision    recall  f1-score   support

       B-LOC      0.882     0.863     0.873      1975
       I-LOC      0.841     0.678     0.750       397
      B-MISC      0.915     0.797     0.852      1062
      I-MISC      0.833     0.593     0.693       344
       B-ORG      0.788     0.751     0.769      1617
       I-ORG      0.786     0.791     0.788      1086
       B-PER      0.827     0.887     0.856      1639
       I-PER      0.867     0.947     0.905      1029

   micro avg      0.842     0.822     0.832      9149
   macro avg      0.842     0.788     0.811      9149
weighted avg      0.842     0.822     0.830      9149
```

```
(metrics.flat_classification_report(

    y_test, y_pred, labels=sorted_labels, digits=3

))
```

You might also want to remove “O” category

```
labels = list(crf.classes_)

labels.remove('O')
```

Question 2.3:

Now that you have done a single model - try using RandomSearch to get the best parameters for our data. Use this param grid:

```
params_space = {

    'c1': scipy.stats.expon(scale=0.5),

    'c2': scipy.stats.expon(scale=0.05),

}
```

And print the best parameter and score.

Hint:

```
rs = RandomSearchCV(crf, params_space,

                        cv=3,

                        verbose=1,

                        n_jobs=-1,

                        n_iter=50,

                        scoring=f1_scorer
```

```
print('best params:', rs.best_params_)

print('best CV score:', rs.best_score_)
```

Question 2.4:

For the best model that you have picked with Grid/Random Search, try getting the lowest transition (bottom 5) features by using `transition_features_` method of the model class. Also, get the bottom-most (5) state feature using `state_features_` method that tells you the worst words for specific NER.

Hint:

```
Top unlikely transitions:
B-PER  -> B-MISC  -2.538594
B-LOC  -> I-MISC  -2.668992
B-ORG  -> I-PER   -2.734719
I-ORG  -> B-PER   -2.772293
I-ORG  -> B-LOC   -2.811853
B-LOC  -> B-PER   -2.848869
B-LOC  -> I-PER   -2.908354
```

```
Top negative:
-1.891320 I-PER    word[-2:]:ho
-1.899537 I-PER    word[-3:]:ion
-1.916480 B-ORG    word.lower():washington
-1.927891 O        word[-2:]:la
-1.939413 I-PER    postag[:2]:VB
-1.988648 O        word[-2:]:TT
-2.084616 I-LOC    +1:word.lower():at
-2.091358 O        word[-3:]:iet
-2.147855 O        +1:word.lower():welfare
```