IDA_Task2_BIONLP

April 24, 2023

```
[ ]:
```

1 BIONLP 2004

This dataset contains abstracts from MEDLINE, a database containing journal articles from fields including medicine and pharmacy. The data was collected by searching for the terms 'human', 'blood cells' and 'transcription factors', and then annotated with five entity types: DNA, protein, cell type, cell line, RNA.

More information in the paper: https://aclanthology.org/W04-1213.pdf

The data can be downloaded from HuggingFace: https://huggingface.co/datasets/tner/bionlp2004

```
[ ]:
```

```
[2]: %load_ext autoreload
%autoreload 2

# Use HuggingFace's datasets library to access the financial_phrasebank dataset
from datasets import load_dataset

import numpy as np
```

```
[3]: dataset = load_dataset(
    "tner/bionlp2004",
    cache_dir='./data_cache'
)

print(f'The dataset is a dictionary with {len(dataset)} splits: \n\n{dataset}')
```

```
Reusing dataset bio_nlp2004 (./data_cache\tner___bio_nlp2004\bionlp2004\1.0.0\9f
41d3f0270b773c2762dee333ae36c29331e2216114a57081f77639fdb5e904)

  0%|          | 0/3 [00:00<?, ?it/s]

The dataset is a dictionary with 3 splits:

DatasetDict({
    train: Dataset({
        features: ['tokens', 'tags'],
        num_rows: 16619
    })
    validation: Dataset({
        features: ['tokens', 'tags'],
        num_rows: 1927
    })
    test: Dataset({
        features: ['tokens', 'tags'],
        num_rows: 3856
    })
})
```

The dataset is already split into train, validation and test. It may be useful to reformat the DatasetDict object into lists of sentences and tags:

[220]:
```
#dataset['train'][1]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[220], line 2
      1 #dataset['train'][1]
----> 2 test_set = DatasetDict["test"]["features"]

NameError: name 'DatasetDict' is not defined
```

[203]:
```
train_sentences_ner = [item['tokens'] for item in dataset['train']]
train_labels_ner = [[str(tag) for tag in item['tags']] for item in␣
 ↪dataset['train']]

val_sentences_ner = [item['tokens'] for item in dataset['validation']]
val_labels_ner = [[str(tag) for tag in item['tags']] for item in␣
 ↪dataset['validation']]

test_sentences_ner = [item['tokens'] for item in dataset['test']]
test_labels_ner = [[str(tag) for tag in item['tags']] for item in␣
 ↪dataset['test']]
```

[5]:
```
print(f'Number of training sentences = {len(train_sentences_ner)}')
print(f'Number of validation sentences = {len(val_sentences_ner)}')
print(f'Number of test sentences = {len(test_sentences_ner)}')
```

```
Number of training sentences = 16619
Number of validation sentences = 1927
Number of test sentences = 3856
```

[6]:
```
print(f'What does one instance look like from the training set?␣
↪\n\n{train_sentences_ner[234]}')
print(f'...and here is its corresponding label \n\n{train_labels_ner[234]}')
```

```
What does one instance look like from the training set?

['Hence', ',', 'PPAR', 'can', 'positively', 'or', 'negatively', 'influence',
'TH', 'action', 'depending', 'on', 'TRE', 'structure', 'and', 'THR', 'isotype',
'.']
…and here is its corresponding label

['0', '0', '3', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0', '3',
'4', '0']
```

[7]:
```
print(f'Number of unique labels: {np.unique(np.concatenate(train_labels_ner))}')
```

```
Number of unique labels: ['0' '1' '10' '2' '3' '4' '5' '6' '7' '8' '9']
```

These are the tags used to annotate the entities:

[8]:
```
# mapping from labels to the tags

id2label = {
    "O": 0,
    "B-DNA": 1,
    "I-DNA": 2,
    "B-protein": 3,
    "I-protein": 4,
    "B-cell_type": 5,
    "I-cell_type": 6,
    "B-cell_line": 7,
    "I-cell_line": 8,
    "B-RNA": 9,
    "I-RNA": 10
}

label2id = {v:k for k, v in id2label.items()}
print(label2id)
```

```
{0: 'O', 1: 'B-DNA', 2: 'I-DNA', 3: 'B-protein', 4: 'I-protein', 5:
'B-cell_type', 6: 'I-cell_type', 7: 'B-cell_line', 8: 'I-cell_line', 9: 'B-RNA',
10: 'I-RNA'}
```

[62]:
```
#train_sentences_ner[0]
train_set = [list(zip(s['tokens'], [id2label[tok] for tok in s['ner_tags']]))␣
↪for s in train_dataset][:-1]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

```
Cell In[62], line 2
      1 #train_sentences_ner[0]
----> 2 train_set = [list(zip(s['tokens'], [id2label[tok] for tok in␣
 ↪s['ner_tags']])) for s in train_dataset][:-1]

NameError: name 'train_dataset' is not defined
```

[143]:
```
# convert label ref from string to label
int_train_labels = []
for sent in train_labels_ner:
    sent = list(map(int, sent))
    int_train_labels.append(sent)
    #print(sent)
```

[212]:
```
# working int to label
train_lab_full = []

for vector in int_train_labels:
    #vector = list(map(int, vector))
    #print(sent)
    sent_labels = []
    for label in vector:
        converted  = label2id[label]
        sent_labels.append(converted)
        #print(converted)
        #res = list(zip(train_sentences_ner, train_labels_ner[idx]))
        #train_set.append(res)
    train_lab_full.append(list(sent_labels))
```

[213]:
```
print(len(train_lab_full[1]))
print(train_lab_full[1])
print(len(train_labels_ner[1]))
print(train_labels_ner[1])
```

```
36
['O', 'B-protein', 'O', 'O', 'O', 'O', 'O', 'O', 'B-protein', 'I-protein', 'O',
'O', 'O', 'O', 'B-protein', 'I-protein', 'O', 'O', 'O', 'O', 'B-protein', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']
36
['0', '3', '0', '0', '0', '0', '0', '0', '3', '4', '0', '0', '0', '0', '3', '4',
'0', '0', '0', '0', '3', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0']
```

[204]:
```
# working zip
train_set = [list(zip(sentence, train_lab_full[idx])) for idx, sentence in␣
 ↪enumerate(train_sentences_ner)]
```

```
[214]: # test_set
       test_set
```

```
[221]: # convert label ref from string to label
       int_test_labels = []
       for sent in test_labels_ner:
           sent = list(map(int, sent))
           int_test_labels.append(sent)
           #print(sent)
```

```
[222]: # working int to label
       test_lab_full = []

       for vector in int_test_labels:
           #vector = list(map(int, vector))
           #print(sent)
           sent_labels = []
           for label in vector:
               converted  = label2id[label]
               sent_labels.append(converted)
               #print(converted)
               #res = list(zip(train_sentences_ner, train_labels_ner[idx]))
               #train_set.append(res)
           test_lab_full.append(list(sent_labels))
```

```
[223]: test_set = [list(zip(sentence, test_lab_full[idx])) for idx, sentence in␣
        ↪enumerate(test_sentences_ner)]
```

```
[225]: #test_set[0]
```

```
[ ]:
```

```
[ ]:
```

```
[215]: #train_sentences_ner
```

```
[ ]:
```

```
[217]: import nltk

       # Train a CRF NER tagger
       def train_CRF_NER_tagger(train_set):
           ### WRITE YOUR OWN CODE HERE
           tagger = nltk.tag.CRFTagger()
           tagger.train(train_set, 'model.crf.tagger')
           return tagger  # return the trained model
```

```
tagger = train_CRF_NER_tagger(train_set)
```

[218]:
```
predicted_tags = tagger.tag_sents(test_sentences_ner)
```

[ ]:
```

```

[226]:
```
def extract_spans(tagged_sents):
    """
    Extract a list of tagged spans for each named entity type,
    where each span is represented by a tuple containing the
    start token and end token indexes.

    returns: a dictionary containing a list of spans for each entity type.
    """
    spans = {}

    for sidx, sent in enumerate(tagged_sents):
        start = -1
        entity_type = None
        for i, (tok, lab) in enumerate(sent):
            if 'B-' in lab:
                start = i
                end = i + 1
                entity_type = lab[2:]
            elif 'I-' in lab:
                end = i + 1
            elif lab == 'O' and start >= 0:

                if entity_type not in spans:
                    spans[entity_type] = []

                spans[entity_type].append((start, end, sidx))
                start = -1
        # Sometimes an I-token is the last token in the sentence, so we still␣
 ↪have to add the span to the list
        if start >= 0:
            if entity_type not in spans:
                spans[entity_type] = []

            spans[entity_type].append((start, end, sidx))

    return spans


def cal_span_level_f1(test_sents, test_sents_with_pred):
    # get a list of spans from the test set labels
    gold_spans = extract_spans(test_sents)
    # get a list of spans predicted by our tagger
    pred_spans = extract_spans(test_sents_with_pred)

    # compute the metrics for each class:
    f1_per_class = []

    ne_types = gold_spans.keys()  # get the list of named entity types (not the␣
 ↪tags)

    for ne_type in ne_types:
        # compute the confusion matrix
        true_pos = 0
        false_pos = 0

        for span in pred_spans[ne_type]:
            if span in gold_spans[ne_type]:
                true_pos += 1
            else:
                false_pos += 1

        false_neg = 0
        for span in gold_spans[ne_type]:
            if span not in pred_spans[ne_type]:
                false_neg += 1

        if true_pos + false_pos == 0:
            precision = 0
        else:
            precision = true_pos / float(true_pos + false_pos)

        if true_pos + false_neg == 0:
            recall = 0
        else:
            recall = true_pos / float(true_pos + false_neg)

        if precision + recall == 0:
            f1 = 0
        else:
            f1 = 2 * precision * recall / (precision + recall)

        f1_per_class.append(f1)
        print(f'F1 score for class {ne_type} = {f1}')

    print(f'Macro-average f1 score = {np.mean(f1_per_class)}')

cal_span_level_f1(test_set, predicted_tags)
```

```
F1 score for class protein = 0.038578411943453565
F1 score for class cell_type = 0
F1 score for class DNA = 0
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[226], line 86
     82         print(f'F1 score for class {ne_type} = {f1}')
     84     print(f'Macro-average f1 score = {np.mean(f1_per_class)}')
---> 86 cal_span_level_f1(test_set, predicted_tags)

Cell In[226], line 55, in cal_span_level_f1(test_sents, test_sents_with_pred)
     52 true_pos = 0
     53 false_pos = 0
---> 55 for span in pred_spans[ne_type]:
     56     if span in gold_spans[ne_type]:
     57         true_pos += 1

KeyError: 'cell_line'
```

```
[ ]:
```