WikiNER:

Brute-force Named Entity Recognition leveraging *Wikipedia* dataset

COM S 579X Natural Language Processing, Fall 2022
December 8, 2022

Mario Mastrandrea, Yuting Yang

IOWA STATE UNIVERSITY

What is **Named Entity Recognition**?

Text

Iowa State University of Science and Technology is a public land-grant research university in Ames, Iowa.

Tags

- [Organization] Iowa State University of Science and Technology
- [Location] Ames
- [Location] Iowa

What is **Named Entity Recognition**?

Text

[Iowa State University of Science and

B I I I I I

Technology]$_{ORG}$ is a public land-grant

I

research university in [Ames]$_{LOC}$,

B

[Iowa]$_{LOC}$.

B

Tagging

Tags

- [Organization] Iowa State University of Science and Technology
- [Location] Ames
- [Location] Iowa

What is **Named Entity Recognition**?

Text

[Iowa State University of Science and
B I I I I I
Technology]ORG is a public land-grant
I
research university in [Ames]LOC,
B
[Iowa]LOC.
B

Tagging

Tags

- [Organization] Entity 1 on *Wikipedia*
- [Location] Entity 2 on *Wikipedia*
- [Location] Entity 2 on *Wikipedia*
- ……

WikiNER uses *Wikipedia* entities as Named Entities tags!

Why WikiNER?

The entities in *Wikipedia* are a good source of tags to recognize WHO, WHERE, and WHAT in a sentence

- **Massive** in the number of entities of each type (i.e., organization, locations, persons)
- Being **evolving** and expanding
- Applicable to **different domains**

Use the **top ranked *Wikipedia* Named Entities** to label a sentence with a **brute-force approach**

How did we build **WikiNER?**

1. Retrieve ***Wikipedia* top ranked NEs** from *Wikidata* API
2. Build a **brute-force model** to perform NER tagging
3. **Evaluate** the model using a tagged *corpus* as a benchmark

Retrieve ***Wikipedia* top ranked NEs** from ***Wikidata* API**

1. download ***.csv* file** from [*Wikidata QRank project*] with **ordered** *Wikipedia* entities

Entity	QRank
Q178995	219893853
Q635	113674399
Q866	93345399
…	…

2. call ***Wikidata* API** for each entity to retrieve its **English label**

*https://www.wikidata.org/w/api.php?action=wbgetentities&ids=**Q178995**&languages=**en**&format=json&props=**labels***

3. create an output ***.csv* file** with the **top k Named Entities labels**

Entity	QRank	Label
Q178995	219893853	**HTTP cookie**
Q635	113674399	**Cleopatra**
Q866	93345399	**YouTube**
…	…	**…**

Retrieve ***Wikipedia*** **top ranked NEs** - with **aliases**

```
def top_N_NEs(
    input_csv_ranking_file_path: str,
    output_csv_file_path: str,
    top_N=1000,
    aliases=False
):
```

The function we implemented to extract the labels

- It is also possible to retrieve all the **aliases** of a Named Entity with the same API, **increasing** the **performances**!

*https://www.wikidata.org/w/api.php?action=wbgetentities&ids=Q178995&languages=en&format=json&props=labels|**aliases***

Entity	QRank	Label
Q178995	219893853	**HTTP cookie**
Q178995	219893853	**cookie**
Q178995	219893853	**web cookie**
Q178995	219893853	**browser cookie**
Q178995	219893853	**internet cookie**
Q178995	219893853	**cookies**
Q635	113674399	**Cleopatra**
Q635	113674399	**Cleopatra VII Philopator**
Q635	113674399	**Cleopatra VII**
Q866	93345399	**YouTube**
Q866	93345399	**YT**
…	…	**…**

Build a **brute-force model** to perform **NER tagging**

```
def brutal_force_NER(
    sentence_tokens: list[str],
    NE_list: list[str],
    tokenizer,
    scheme="BIO"
):
```

The function we implemented to tag a sentence using the Wikipedia NEs with brute-force approach

1. Find all the **not-overlapping *Matches*** in the sentence using all the NEs (brute-force)
2. Represent this matches according to the specified **tagging scheme** (i.e., *BIO*, *BILOU*)

*A *Match* is an **occurrence** of a Named Entity in the sentence, represented as a tuple of **indexes** *(a,b)*: *a* is the **start** index, *b* is the (exclusive) **ending** index

Example of **brute-force tagging**

sentence_tokens = [‘I’, ‘went', ‘to’, ‘school’, ‘at’, ‘Iowa’, ‘State’, ‘University’, ‘in’, ‘Fall’, ‘2022’]

NE_list = ['Microsoft', 'Iowa State', 'Fall 2022', 'Deep Learning', 'Iowa State University', '2022', 'school']

(3,4), (5,8), (9,11)

Matches

I went to school at Iowa State University in Fall 2022

O O O B O B I I O B I

O O O U O B I L O B I

BIO and BILOU representations

Challenge #1: **Overlapping NEs**

- We solved the problem of **overlapping** Named Entities prioritizing always the **longest Named Entity**

sentence_tokens = [‘I’, ‘went', ‘to’, ‘school’, ‘at’, ‘Iowa’, ‘State’, ‘University’, ‘in’, ‘Fall’, ‘2022’]
NE_list = ['Microsoft', 'Iowa State', 'Fall 2022', 'Deep Learning', 'Iowa State University', '2022', 'school']

(3,4), (5,7), (5,8)
(9,11), (10,11)

overlapping *Matches*

(3,4), (5,8), (9,11)

not-overlapping *Matches*

I went to school at Iowa State University in Fall 2022
O O O O O B I I O B I

BIO representation

Supported **tagging schemes**

- Our ***WikiNER*** model supports 2 different tagging schemes:

BIO:

B – Beginning
I – Inside
O – Outside

BILOU:

B – Beginning
I – Inside
L – Last
O – Outside
U – Unit

Challenge #2: **Entity type (LOC, PER, ORG, MISC, etc.)**

- NER problem also expects to find the **type** of each Named Entity (LOCation, PERson, ORGanization, etc.)

- But this is **impossible** using just a the *Wikidata* API, since it does **not** provide the type of an entity

- **Solution**: using another model (like **BERT**) to also find the type of NEs (*to be continued*)

Evaluate the model using a tagged *corpus* as **benchmark**

- Use **CoNLL2003 dataset** as benchmark
  - It contains a *corpus* of sentences whose Named Entities have been tagged in their respective **types** using the **BIO scheme**
- Use the ***WikiNER*** model to tag its sentences
- Evaluate the **performances** comparing the **predicted tags** with the **real tags** in the *corpus* (ignoring the NE type)

token	POS tag	chunk tag	NER tag
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**
Peter	NNP	B-NP	**B-PER**
…	…	…	**…**

Evaluation metrics

- **Precision**

$$\frac{True\ Positives}{True\ Positives + False\ Positives}$$

Among all the predicted NEs, how many of them were **true NEs**

- **Recall**

$$\frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Among all the real NEs, how many of them we were able to **predict**

- **F1-score**

$$2 * \frac{Precision\ * Recall}{Precision + Recall}$$

Harmonic mean between *precision* and *recall*

WikiNER performances on *CoNLL2003* dataset

- We used ***seqeval*** library to compute *precision*, *recall* and *F1-score*
- **Compare** the results using **different numbers** of Wikipedia top NEs

#NEs	Precision	Recall	F1-score
100	0.94	0.05	0.09
1,000	0.89	0.14	0.24
10,000	0.26	0.22	0.24

*with **no** aliases*

#NEs	Precision	Recall	F1-score
100	0.82	0.06	0.11
1,000	0.44	0.17	**0.25**
10,000	0.15	**0.29**	0.20

***with** aliases*

Timing problems

- *WikiNER* is a very **time-consuming** approach!
  - **extracting** *Wikipedia* NEs
  - **executing** the brute-force model

#NEs	time
100	~20s
1,000	~3min
10,000	~30min

*estimated times to **extract** NEs*

#NEs	no aliases	with aliases
100	~5min	~20min
1,000	~30min	~2h
10,000	~3h	~12h

*estimated times to execute **WikiNER** on **CoNLL2003***

Use A BERT Model to Perform the NER Tagging

- Due to the disadvantages of the brute-force model (i.e., lack of entity type and time-consuming), we used a pretrained BERT model to perform the task on CONLL2003.
- This task takes four steps
  - Step1: Create tokenized dataset based on CONLL2003
  - Step2: Create input objects for Trainer, including tokenizer, evaluation metrics, and data collator
  - Step3: Define model and arguments
  - Step4: Train the model

Step 1: Create Tokenized Dataset

- Issue 1: The NER tags for the tokens in CONLL2003 were designated as integers rather than label names

```
Tokens:       EU  rejects German call to boycott British lamb .
NER tag no.:  3   0       7      0    0  0       7       0    0
```

- Solution: replace integer ids with label names for tokens

```
{'O': 0,
 'B-PER': 1,
 'I-PER': 2,
 'B-ORG': 3,
 'I-ORG': 4,
 'B-LOC': 5,
 'I-LOC': 6,
 'B-MISC': 7,
 'I-MISC': 8}
```

```
Tokens:       EU    rejects German call to boycott British lamb .
NER tag no.:  3     0       7      0    0  0       7       0    0
NER tag:      B-ORG 0       B-MISC 0    0  0       B-MISC  0    0
```

Step 1: Create Tokenized Dataset

- Issue 2: The tokens in CONLL2003 need to be tokenized in a way that fits the BERT model
  - “None” as word id
- Solution
  - Replace -100 for “None” as word id

```
inputs = tokenizer(dataset['train'][0]['tokens'], is_split_into_words=True)
print(inputs.tokens())
print(inputs.word_ids())
```

✓ 0.5s

```
['[CLS]', 'EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'la', '##mb', '.', '[SEP]']
[None, 0, 1, 2, 3, 4, 5, 6, 7, 7, 8, None]
```

Step 1: Create Tokenized Dataset

- Taken together, the tokenized dataset was created in the following three steps

```python
def align_labels_with_tokens(labels, word_ids):
    new_labels = []
    current_word = None
    for word_id in word_ids:
        if word_id != current_word:
            current_word = word_id
            label = -100 if word_id is None else labels[word_id]
            new_labels.append(label)
        elif word_id is None:
            new_labels.append(-100)
        else:
            label = labels[word_id]

            # print(f'before: {label}')
            if label % 2 == 1:
                label += 1
            # print(f'after: {label}')
            new_labels.append(label)
    return new_labels
```

```python
def tokenize_and_align_labels(examples):
    tokenzied_inputs = tokenizer(
        examples['tokens'], truncation = True, is_split_into_words = True
    )
    all_labels = examples['ner_tags']
    new_labels = []
    for i, labels in enumerate(all_labels):
        word_ids = tokenzied_inputs.word_ids(i)
        new_labels.append(align_labels_with_tokens(labels, word_ids))
    tokenzied_inputs['labels'] = new_labels
    return tokenzied_inputs
```

```python
tokenized_dataset = dataset.map(
    tokenize_and_align_labels,
    batched = True,
    remove_columns = dataset['train'].column_names
)
```

Step 2: Create Other Objects for Trainer

- Tokenizer

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
```

- Evaluation metrics

```
def compute_metrics(eval_preds):
    logits, labels = eval_preds
    predictions = np.argmax(logits, axis=2)

    true_labels = [[label_names[l] for l in label if l != -100] for label in labels]
    true_predictions = [
        [label_names[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    all_metrics = metric.compute(predictions=true_predictions, references=true_labels)
    return {
        "precision": all_metrics["overall_precision"],
        "recall": all_metrics["overall_recall"],
        "f1": all_metrics["overall_f1"],
        "accuracy": all_metrics["overall_accuracy"],
}
```

- Data collator

```
from transformers import DataCollatorForTokenClassification
data_collator = DataCollatorForTokenClassification(tokenizer = tokenizer)
```

Step 3: Define Model and Training Arguments

- Pre-trained BERT model

```
from transformers import AutoModelForTokenClassification
model = AutoModelForTokenClassification.from_pretrained(
        'bert-base-cased',
        num_labels = 9,
)
```

- Training model arguments

```
from transformers import TrainingArguments
args = TrainingArguments(
    'bert-finetuned-ner',
    evaluation_strategy = 'epoch',
    # save_strategy = 'epoch',
    per_device_train_batch_size= 16,
    per_device_eval_batch_size= 16,
    learning_rate = 2e-5,
    num_train_epochs = 3,
    weight_decay = 0.01,
)
```

Step 4: Train the BERT Model

- Put the objects created in the previous three steps together and create the following trainer. Train the model.

```python
from transformers import Trainer
trainer = Trainer(
    model = model,
    args = args,
    train_dataset = tokenized_dataset['train'],
    eval_dataset = tokenized_dataset['validation'],
    data_collator = data_collator,
    compute_metrics = compute_metrics,
    tokenizer = tokenizer,
)
```

```python
trainer.train()
```

Evaluate the Training Results of the BERT Model

- Issue 3: Low f1 score and accuracy
  - *compute_metrics* function: *axis = -1*
  - *args: num_train_epochs = 1*

Epoch	Eval Loss	Eval Precision	Eval Recall	Eval F1	Eval Accuracy
1	0.39	0.46	0.40	0.43	0.88

- Solution: Try other configurations

Evaluate the Training Results of the BERT Model

- New configuration
  - *compute_metrics* function: *axis = 2*
  - *args: num_train_epochs = 3;* add *per_device_train_batch_size = 16* and *per_device_eval_batch_size = 16*

Epoch	Eval Loss	Eval Precision	Eval Recall	Eval F1	Eval Accuracy
1	0.0730	0.9041	0.9262	0.9150	0.9800
2	0.0599	0.9230	0.9461	0.9344	0.9848
3	0.0566	0.9237	0.9478	0.9356	0.9857

Conclusions

- *WikiNER* brutal force model is **ineffective** to perform a NER task
  - **Low performances** (max recall: 29% - 10,000 NEs and aliases)
  - **Time-consuming** (it takes hours)
  - **No NEs types**

- Two issues in *Wikipedia* entities:
  - The presence of **more NEs** naturally **decreases precision** (although **increases recall**)
  - The *Wikipedia* entities are **not universal** enough to capture benchmark entities

- Given proper configurations in *TrainingArguments()*, BERT model performs notably better than the brute force model
  - The greater the batch size, the greater the performance

Questions?

Thank You!