Complex Named Entity Recognition

By: Eric Tieu

Named Entity Recognition(NER) is a sub-task in Natural Language Processing(NLP) which identifies entities in text normally of types Person(PER), Organisation(ORG), and Location(LOC). There are a few methods which can be used to identify these entities. Regular expressions use the format of text to identify entities such as locations. For example a street is usually formatted as the street name followed by ‘Street’ or ‘St.’. This allows us to identify addresses when text is formatted in the following way: [number] [street name] [suffix], where the suffix is from a chosen list including things such as ‘Ave’, ‘Crescent’, or ‘St’. Another form of NER is Gazetteers which are lists of known named entities(NE). Gazetteers are more suited for a specific collection of words within their own system. For example if we were to use gazetteers to identify every entity ever mentioned in every language our list would be longer than a list of all the words of every language.

The conventional approach to NER today is using machine learning to train Large Language Models(LLMs), more specifically transformer models. These models use an encoder to process the input and a decoder to produce a prediction. It tokenizes inputs, captures semantic meanings, and discovers relationships between these tokens through mathematical equations. Once the model has discovered these relationships it can detect patterns in text similar to the patterns present in text the model was trained on.

For these models IOB tagging is conventionally used for formatting the prediction output. It is in the form of chunks, chunks being NEs. ‘B’ represents the beginning of a chunk, ‘I’ represents the inside of a chunk, and ‘O’ represents the outside of a chunk and not a NE. So a sentence such as ‘The book Lord of the Flies is a book’ would be tagged as so:

the	O
book	O
lord	B-[entity type]
of	I-[entity type]
the	I-[entity type]
flies	I-[entity type]
is	O
a	O
book	O

Normally a LLM is trained to tag the most basic entity types, PER, ORG, LOC. The goal of Complex NER, is to be able to identify things such as titles of creative works which consist of multiple words. The patterns complex NEs introduce vary greatly from the patterns found with traditional NEs. In this project we aim to train a LLM to be able to identify such Complex NEs. We also aim to identify entities more specifically, so rather than being able to identify approximately six unique entity types, the model trained in this project should be able to identify dozens of unique entity types based on their associated patterns. Since we’re using a LLM all we need to do is preprocessing, and specifying the tags that the model will be trying to predict. The pattern recognition will be done automatically by the model.

The data set used in this project is the dataset used in ‘SemEval 2023 Task 2: MultiCoNER II Multilingual Complex Named Entity Recognition’ with 33 unique NE types. This dataset can be used to finetune a number of LLMs, however for this project I opted to use the dataset to finetune a model of bert-base-uncased as the data that is provided in the SemEval task is already uncased like our model. Using a cased dataset and model could improve the accuracy; ‘cased’ referring to uppercase and lowercase. The dataset is multilingual, however this project was done using only the english data since bert-base-uncased is only trained on english data. However, since the format of the data is the same for each language, the preprocessing and training should not have any issues if code is added to specify the language for each data entry and a multilingual LLM is used, such as XLM-RoBERTa.

The code includes a number of dependencies, such as pytorch, cuda, transformers(from huggingface.co)

First the data needs to be preprocessed so that it will work with our model. The dataset chosen comes in conll format which can also be formatted in many ways. The dataset includes an id for each entry as well as the language domain, as seen on the top row of *Figure 1*. However, since we’re only training our model with the english data this row has to be dropped from the data. We can also see there are 2 blank columns in our data represented by the underscores. These can represent part of speech tagging, lemmatized words, semantics, etc. as seen in *Figure 2*. Our data has excluded these tags as our model will process the words themselves and produce these if required. The only two entries we care about is the word and the respective tag.

```
# id 309f5b26-951e-472b-948e-47632249862b	domain=en
robert _ _ B-OtherPER
gottschalk _ _ I-OtherPER
1939 _ _ O
academy _ _ B-VisualWork
award _ _ I-VisualWork
winner _ _ O
and _ _ O
founder _ _ O
of _ _ O
panavision _ _ B-ORG
```

Figure 1: Example entry of training data

```
1	Då	då	ADV	AB	_
2	var	vara	VERB	VB.PRET.ACT	Tense=Past|Voice=Act
3	han	han	PRON	PN.UTR.SIN.DEF.NOM	Case=Nom|Definite=Def|Gender=Com|Number=Sing
4	elva	elva	NUM	RG.NOM	Case=Nom|NumType=Card
5	år	år	NOUN	NN.NEU.PLU.IND.NOM	Case=Nom|Definite=Ind|Gender=Neut|Number=Plur
6	.	.	PUNCT	DL.MAD	_
```

Figure 2: Example of morphological annotation in conll format

First we have to import our data as pandas format. To do this we define our column names, TOKEN, POS, CHUNK, NE. We also create a new column for sentence number for later use; an example of our data in pandas format is shown in *Figure 4*. We only need to use the words(TOKEN) and tags(NE) so we remove the Part of Speech(POS) and CHUNK columns. For our model we'll need our data in the form of token lists(Sentence) and tags(IOBTags) as shown in *Figure 5*. For each token we add a new column, a sentence list which is a concatenation of TOKENs with matching sentence numbers. We repeat this for the NE column. Since we only need one list for each sentence we reform the data, dropping duplicates. *Figure 3* shows the code used to achieve these steps.

```
###Data Preprossesing
##function to convert conll format to pandas
def read_conll(filename):
    df = pd.read_csv(filename,
                    na_values=['#'],
                    sep = ' ', header = None, keep_default_na = False,
                    names = ['TOKEN', 'POS', 'CHUNK', 'NE'],
                    quoting = 3, skip_blank_lines = False)
    df = df.dropna()
    df['SENTENCE'] = (df.TOKEN == '').cumsum()
    return df[df.TOKEN != '']
##use only TOKEN NE SENTENCE
def preprossesingData(data):
    data = data[["TOKEN", "NE", "SENTENCE"]].drop_duplicates().reset_index(drop=True)
    data['Sentence'] = data[['SENTENCE','TOKEN','NE']].groupby(['SENTENCE'])['TOKEN'].transform(lambda x: ' '.join(x))
    data['IOBTags'] = data[['SENTENCE','TOKEN','NE']].groupby(['SENTENCE'])['NE'].transform(lambda x: ','.join(x))
    #reform the data as sentences(tokenlists) and IOBTag lists
    data = data[["Sentence", "IOBTags"]].drop_duplicates().reset_index(drop=True)
    return data
###Data Preprocessing
```

Figure 3: Data preprocessing code

	TOKEN	POS	CHUNK	NE	SENTENCE
2	robert	_	_	B-OtherPER	1
3	gottschalk	_	_	I-OtherPER	1
4	1939	_	_	O	1
5	academy	_	_	B-VisualWork	1
6	award	_	_	I-VisualWork	1

Figure 4: Format of our data read in by the read_conll function

```
robert gottschalk 1939 academy award winner and founder of panavision
B-OtherPER,I-OtherPER,O,B-VisualWork,I-VisualWork,O,O,O,O,B-ORG
```

Figure 5: Format of our data after preprocessingData function

To define our model we need to extract the tags from the dataset. We do this by iterating over the training data and adding a tag to a list if a new unique tag is found. The resulting tags from *Figure 6* are shown in *Figure 7*.

```
id2label = {v: k for v, k in enumerate(trainingData.NE.unique())}
label2id = {k: v for v, k in enumerate(trainingData.NE.unique())}
#print(label2id)
```

Figure 6: Code for listing tags/labels

```
{'B-OtherPER': 0, 'I-OtherPER': 1, 'O': 2, 'B-VisualWork': 3, 'I-VisualWork': 4, 'B-ORG': 5, 'B-Artist': 6, 'I-Artist': 7, 'B-HumanSettlement': 8, 'B-WrittenWork': 9, 'B-Software': 10,
 'I-Software': 11, 'I-WrittenWork': 12, 'B-Politician': 13, 'I-Politician': 14, 'B-Athlete': 15, 'I-Athlete': 16, 'B-MusicalWork': 17, 'I-MusicalWork': 18, 'I-HumanSettlement': 19, 'B-
Facility': 20, 'I-Facility': 21, 'B-Scientist': 22, 'I-Scientist': 23, 'B-Cleric': 24, 'I-Cleric': 25, 'I-ORG': 26, 'B-SportsGRP': 27, 'B-MusicalGRP': 28, 'I-MusicalGRP': 29, 'B-Sports
Manager': 30, 'I-SportsManager': 31, 'B-PublicCorp': 32, 'I-PublicCorp': 33, 'B-OtherPROD': 34, 'B-MedicalProcedure': 35, 'I-MedicalProcedure': 36, 'B-ArtWork': 37, 'I-ArtWork': 38, 'B
-Food': 39, 'I-Food': 40, 'B-Station': 41, 'I-Station': 42, 'I-OtherPROD': 43, 'B-CarManufacturer': 44, 'B-OtherLOC': 45, 'I-OtherLOC': 46, 'B-PrivateCorp': 47, 'I-SportsGRP': 48, 'B-D
isease': 49, 'B-Vehicle': 50, 'I-Vehicle': 51, 'I-PrivateCorp': 52, 'B-Medication/Vaccine': 53, 'B-Symptom': 54, 'I-Medication/Vaccine': 55, 'I-Disease': 56, 'B-AnatomicalStructure': 5
7, 'I-AnatomicalStructure': 58, 'I-Symptom': 59, 'B-AerospaceManufacturer': 60, 'I-CarManufacturer': 61, 'B-Drink': 62, 'I-Drink': 63, 'B-Clothing': 64, 'I-AerospaceManufacturer': 65,
'I-Clothing': 66}
```

Figure 7: IOB tags extracted from our dataset

Once extracted the tags from our dataset we can define the model

```python
###Define Model using pretrained model weight, additionally we specify the labels, extracted previously.
model = BertForTokenClassification.from_pretrained('bert-base-uncased',
                                                   num_labels=len(id2label),
                                                   id2label=id2label,
                                                   label2id=label2id)
model.to(device)
###Define Model
```

Figure 8: Define model using the extracted tags

At this point no new code was created to process our data. Parameters are defined, such as MAX_LEN, TRAIN_BATCH_SIZE, EPOCHS, etc. A tokenizer is defined, in our case "BertTokenizer.from_pretrained('bert-base-uncased')" and a function is defined to process our 'Sentence' list and 'IOBTags' list into a trainable dataset. Then we train the model on our dataset.

We can see in *Figure 9* I attempt to use the bert-base-uncased model to do NER. The model was not trained on a dataset with IOB tagging and is unable to produce anything usable. The IOB tag would be where 'LABEL_#' is.

```python
example1untrained.py > ...
from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModelForTokenClassification.from_pretrained("bert-base-uncased")
nlp = pipeline("ner", model=model, tokenizer=tokenizer)
example = "Star Wars is a movie that takes place a long time ago in a galaxy far far away… It is about a war in the stars. It stars Mark Hamill."
ner_results = nlp(example)
print(ner_results)
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                    powershell

  File "example2.py", line 3, in <module>
(.env) PS C:\Users\Eric\Desktop\NER> python example1untrained.py
Some weights of BertForTokenClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
[{'entity': 'LABEL_0', 'score': 0.5166005, 'index': 1, 'word': 'star', 'start': 0, 'end': 4}, {'entity': 'LABEL_1', 'score': 0.5123163, 'index': 2, 'word': 'wars', 'start': 5, 'end': 9
}, {'entity': 'LABEL_1', 'score': 0.501504, 'index': 3, 'word': 'is', 'start': 10, 'end': 12}, {'entity': 'LABEL_1', 'score': 0.50921357, 'index': 4, 'word': 'a', 'start': 13, 'end': 1
4}, {'entity': 'LABEL_0', 'score': 0.51538026, 'index': 5, 'word': 'movie', 'start': 15, 'end': 20}, {'entity': 'LABEL_0', 'score': 0.5536463, 'index': 6, 'word': 'that', 'start': 21,
 'end': 25}, {'entity': 'LABEL_0', 'score': 0.5177535, 'index': 7, 'word': 'takes', 'start': 26, 'end': 31}, {'entity': 'LABEL_0', 'score': 0.5858035, 'index': 8, 'word': 'place', 'star
 'end': 25}, {'entity': 'LABEL_0', 'score': 0.5177535, 'index': 7, 'word': 'takes', 'start': 26, 'end': 31}, {'entity': 'LABEL_0', 'score': 0.5858035, 'index': 8, 'word': 'place', 'star
t': 32, 'end': 37}, {'entity': 'LABEL_1', 'score': 0.52281266, 'index': 9, 'word': 'a', 'start': 38, 'end': 39}, {'entity': 'LABEL_0', 'score': 0.55991745, 'index': 10, 'word': 'long',
 'start': 40, 'end': 44}, {'entity': 'LABEL_0', 'score': 0.5345309, 'index': 11, 'word': 'time', 'start': 45, 'end': 49}, {'entity': 'LABEL_1', 'score': 0.51870435, 'index': 12, 'word'
: 'ago', 'start': 50, 'end': 53}, {'entity': 'LABEL_0', 'score': 0.51952404, 'index': 13, 'word': 'in', 'start': 54, 'end': 56}, {'entity': 'LABEL_0', 'score': 0.5069327, 'index': 14,
'word': 'a', 'start': 57, 'end': 58}, {'entity': 'LABEL_1', 'score': 0.5936795, 'index': 15, 'word': 'galaxy', 'start': 59, 'end': 65}, {'entity': 'LABEL_1', 'score': 0.54907876, 'inde
x': 16, 'word': 'far', 'start': 66, 'end': 69}, {'entity': 'LABEL_1', 'score': 0.5344123, 'index': 17, 'word': 'far', 'start': 70, 'end': 73}, {'entity': 'LABEL_1', 'score': 0.5426377,
 'index': 18, 'word': 'away', 'start': 74, 'end': 78}, {'entity': 'LABEL_0', 'score': 0.53972626, 'index': 19, 'word': '…', 'start': 78, 'end': 79}, {'entity': 'LABEL_0', 'score': 0.54
2398, 'index': 20, 'word': 'it', 'start': 80, 'end': 82}, {'entity': 'LABEL_1', 'score': 0.54193795, 'index': 21, 'word': 'is', 'start': 83, 'end': 85}, {'entity': 'LABEL_1', 'score':
0.6338421, 'index': 22, 'word': 'about', 'start': 86, 'end': 91}, {'entity': 'LABEL_0', 'score': 0.5346335, 'index': 23, 'word': 'a', 'start': 92, 'end': 93}, {'entity': 'LABEL_1', 'sc
ore': 0.61937565, 'index': 24, 'word': 'war', 'start': 94, 'end': 97}, {'entity': 'LABEL_1', 'score': 0.6702705, 'index': 25, 'word': 'in', 'start': 98, 'end': 100}, {'entity': 'LABEL_
1', 'score': 0.6138141, 'index': 26, 'word': 'the', 'start': 101, 'end': 104}, {'entity': 'LABEL_1', 'score': 0.5608047, 'index': 27, 'word': 'stars', 'start': 105, 'end': 110}, {'enti
ty': 'LABEL_1', 'score': 0.53088415, 'index': 28, 'word': '.', 'start': 110, 'end': 111}, {'entity': 'LABEL_0', 'score': 0.5372008, 'index': 29, 'word': 'it', 'start': 112, 'end': 114}
, {'entity': 'LABEL_0', 'score': 0.5099443, 'index': 30, 'word': 'stars', 'start': 115, 'end': 120}, {'entity': 'LABEL_1', 'score': 0.5174514, 'index': 31, 'word': 'mark', 'start': 121
, 'end': 125}, {'entity': 'LABEL_1', 'score': 0.57464474, 'index': 32, 'word': 'ham', 'start': 126, 'end': 129}, {'entity': 'LABEL_1', 'score': 0.54363173, 'index': 33, 'word': '##ill'
, 'start': 129, 'end': 132}, {'entity': 'LABEL_1', 'score': 0.5186775, 'index': 34, 'word': '.', 'start': 132, 'end': 133}]
(.env) PS C:\Users\Eric\Desktop\NER>
```

Figure 9: Attempt of using untrained bert-base-uncased for NER

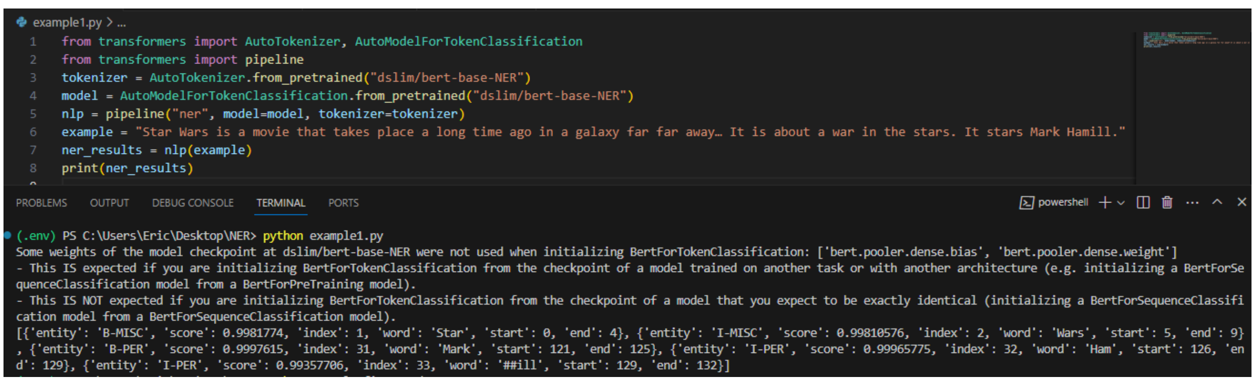
I also tested a bert-base-cased model fine tuned for regular NER, also known as bert-base-NER to compare it to our model trained on a dataset for complex NER. The first thing I tested the models on was a regular sentence *"Star Wars is a movie that takes place a long time ago in a galaxy far far away… It is about a war in the stars. It stars Mark Hamill."*. As we can see from *Figure 10* the regular NER model it classifies Star Wars and Mark Hamill as such:

Star B-MISC
Wars I-MISC
Mark B-PER
Ham I-PER
#ill I-PER

While our model from *Figure 11* gives them more specific tags, but incorrectly tags 'stars':

star B-VisualWork
wars I-VisualWork
stars I-WrittenWork*
mark B-Artist
ham I-Artist
#ill I-Artist

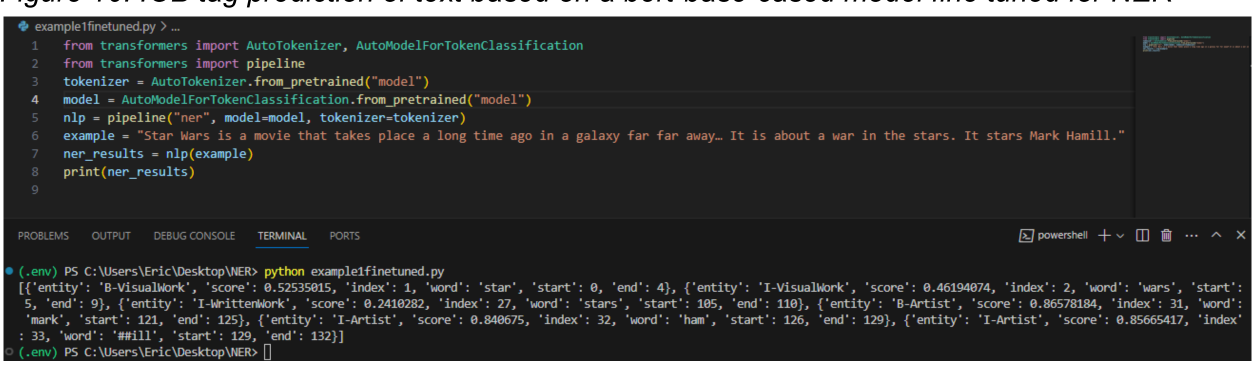
We can see the difference in using a cased model vs an uncased model, the regular NER model using bert-based-cased returns capitalized words, while ours using bert-based-uncased returns each word uncased. This casing is also likely why our model incorrectly predicted 'stars' to be a written work as the model tokenizes and uncases the words before entering the encoder causing the model to ignore casing when producing predictions.



```
example1.py > ...
from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline
tokenizer = AutoTokenizer.from_pretrained("dslim/bert-base-NER")
model = AutoModelForTokenClassification.from_pretrained("dslim/bert-base-NER")
nlp = pipeline("ner", model=model, tokenizer=tokenizer)
example = "Star Wars is a movie that takes place a long time ago in a galaxy far far away… It is about a war in the stars. It stars Mark Hamill."
ner_results = nlp(example)
print(ner_results)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                powershell
(.env) PS C:\Users\Eric\Desktop\NER> python example1.py
Some weights of the model checkpoint at dslim/bert-base-NER were not used when initializing BertForTokenClassification: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
[{'entity': 'B-MISC', 'score': 0.9981774, 'index': 1, 'word': 'Star', 'start': 0, 'end': 4}, {'entity': 'I-MISC', 'score': 0.99810576, 'index': 2, 'word': 'Wars', 'start': 5, 'end': 9}, {'entity': 'B-PER', 'score': 0.9997615, 'index': 31, 'word': 'Mark', 'start': 121, 'end': 125}, {'entity': 'I-PER', 'score': 0.99965775, 'index': 32, 'word': 'Ham', 'start': 126, 'end': 129}, {'entity': 'I-PER', 'score': 0.99357706, 'index': 33, 'word': '##ill', 'start': 129, 'end': 132}]
```

Figure 10: IOB tag prediction of text based on a bert-base-cased model fine tuned for NER



```
example1finetuned.py > ...
from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline
tokenizer = AutoTokenizer.from_pretrained("model")
model = AutoModelForTokenClassification.from_pretrained("model")
nlp = pipeline("ner", model=model, tokenizer=tokenizer)
example = "Star Wars is a movie that takes place a long time ago in a galaxy far far away… It is about a war in the stars. It stars Mark Hamill."
ner_results = nlp(example)
print(ner_results)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                powershell
(.env) PS C:\Users\Eric\Desktop\NER> python example1finetuned.py
[{'entity': 'B-VisualWork', 'score': 0.52535015, 'index': 1, 'word': 'star', 'start': 0, 'end': 4}, {'entity': 'I-VisualWork', 'score': 0.46194074, 'index': 2, 'word': 'wars', 'start': 5, 'end': 9}, {'entity': 'I-WrittenWork', 'score': 0.2410282, 'index': 27, 'word': 'stars', 'start': 105, 'end': 110}, {'entity': 'B-Artist', 'score': 0.86578184, 'index': 31, 'word': 'mark', 'start': 121, 'end': 125}, {'entity': 'I-Artist', 'score': 0.840675, 'index': 32, 'word': 'ham', 'start': 126, 'end': 129}, {'entity': 'I-Artist', 'score': 0.85665417, 'index': 33, 'word': '##ill', 'start': 129, 'end': 132}]
(.env) PS C:\Users\Eric\Desktop\NER>
```

Figure 11: IOB tag prediction of text based on our bert-base-uncased model fine tuned for complex NER using our dataset

The next thing I tested the models on was simply the title of a book with no context "The Catcher in the Rye" as we can see in *Figure 12* after running example2finetuned.py and example2.py respectively our complex NER model is unable to identify any entities, while the regular NER model is only able tag Rye as a location. This example presents the importance of context when attempting NER.

```
(.env) PS C:\Users\Eric\Desktop\NER> python example2finetuned.py
[]
(.env) PS C:\Users\Eric\Desktop\NER> python example2.py
Some weights of the model checkpoint at dslim/bert-base-NER were not used when initializing BertForTokenClassification: ['bert.pooler.dense.weight', 'bert.pooler.dense.bias']
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
[{'entity': 'B-LOC', 'score': 0.7934266, 'index': 6, 'word': 'R', 'start': 19, 'end': 20}, {'entity': 'I-LOC', 'score': 0.5994597, 'index': 7, 'word': '##ye', 'start': 20, 'end': 22}]
(.env) PS C:\Users\Eric\Desktop\NER>
```

Figure 12: Comparison of "The Catcher in the Rye" on bert-base-NER and our model

The final thing I tested was a name with a long title, also without context. As shown in *Figure 13* neither model was able to completely identify the title and name as a single entity. The regular NER shown after running example3.py is able to identify 'Justice Scott Baker' as a 'PER' while the complex NER identifies 'Lord Justice Scott' as a 'OtherPER' and 'Politician' simultaneously, showing that even if the 'beginning' of an entity is tagged one thing that the 'inside' is not. Whether this is a good thing or not, it's the result of using machine learning without specifying rules.

```
example = "Lord Justice Scott Baker"
ner_results = nlp(example)
print(ner_results)

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                powershell

(.env) PS C:\Users\Eric\Desktop\NER> python example3.py
Some weights of the model checkpoint at dslim/bert-base-NER were not used when initializing BertForTokenClassification: ['bert.pooler.dense.weight', 'bert.pooler.dense.bias']
- This IS expected if you are initializing BertForTokenClassification from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForTokenClassification from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
[{'entity': 'B-PER', 'score': 0.9324506, 'index': 2, 'word': 'Justice', 'start': 5, 'end': 12}, {'entity': 'I-PER', 'score': 0.9412371, 'index': 3, 'word': 'Scott', 'start': 13, 'end': 18}, {'entity': 'I-PER', 'score': 0.9988524, 'index': 4, 'word': 'Baker', 'start': 19, 'end': 24}]
(.env) PS C:\Users\Eric\Desktop\NER> python example3finetuned.py
[{'entity': 'B-OtherPER', 'score': 0.42303854, 'index': 1, 'word': 'lord', 'start': 0, 'end': 4}, {'entity': 'I-Politician', 'score': 0.31965846, 'index': 2, 'word': 'justice', 'start': 5, 'end': 12}, {'entity': 'I-Politician', 'score': 0.40261066, 'index': 3, 'word': 'scott', 'start': 13, 'end': 18}]
(.env) PS C:\Users\Eric\Desktop\NER>
```

Figure 13: Comparison of "Lord Justice Scott Baker" on bert-base-NER and our model

The accuracy of the model when evaluated was lower than expected, as seen in *Figure 14* with an average precision of 0.50, recall of 0.56, and f1 of 0.52. I thought this could be an issue with the training data or test data. To figure out whether this was an issue with the training data I had the unique tags and their counts printed out in *Figure 15* to see if the low scores corresponded to low tag counts. However, this was not the case, as the 'Scientist' tag with one of the highest counts had a score of zero on everything. This led me to test some of the test data with tags that got low scores to figure out the issue. It seemed the issue was that some tags were being overruled by certain tags, such as 'PublicCorp' or 'Politician' as demonstrated with the 'AerospaceManufacturer' tag in *Figure 16* and the 'Scientist' tag in *Figure 17.*

We know transformers predict tags based on mathematical equations derived from the relationship between words. So, the behavior of some tags being overridden is understandable. In the example of 'AerospaceManufacturer' it is technically a 'PublicCorp'. However the same does not apply to the 'Scientist' tag, it's unlikely that the context in every single case matches closely enough to another tag that it would result in a score of zero, especially given the prominence of the scientist tag. Low scores which match low counts however do make sense seen with the ‘Drink’ tag. This would be because there would be too few entries to accurately determine the relationships between the tags and other parts of the sentence.

```
Validation Loss: 0.0683729074339198
Validation Accuracy: 0.8774138116365933
C:\Users\Eric\Desktop\NER\.env\lib\site-packages\seqeval\metrics\v1.py:57: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicte
d samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
                       precision    recall  f1-score   support

AerospaceManufacturer       0.26      0.18      0.21      1990
  AnatomicalStructure       0.56      0.51      0.54     13959
              ArtWork       0.09      0.17      0.12      1958
               Artist       0.55      0.86      0.67     84616
              Athlete       0.66      0.83      0.74     41184
       CarManufacturer       0.50      0.20      0.28      5372
               Cleric       0.15      0.02      0.03      7724
             Clothing       0.41      0.39      0.40      4677
              Disease       0.35      0.60      0.44     14833
                Drink       0.36      0.11      0.17      4199
             Facility       0.43      0.59      0.49     29250
                 Food       0.41      0.37      0.39     10613
      HumanSettlement       0.76      0.77      0.77     70560
     MedicalProcedure       0.35      0.51      0.42      9572
   Medication/Vaccine       0.71      0.59      0.64     17365
           MusicalGRP       0.55      0.53      0.54     21952
          MusicalWork       0.54      0.60      0.57     23115
                  ORG       0.42      0.42      0.42     35597
             OtherLOC       0.01      0.01      0.01      7997
             OtherPER       0.30      0.24      0.26     35052
            OtherPROD       0.30      0.34      0.32     22389
           Politician       0.42      0.46      0.44     25829
          PrivateCorp       0.00      0.00      0.00      1443
           PublicCorp       0.34      0.44      0.39     12955
            Scientist       0.00      0.00      0.00      7211
             Software       0.55      0.49      0.52     15870
            SportsGRP       0.73      0.76      0.75     19849
        SportsManager       0.00      0.00      0.00      7732
              Station       0.66      0.62      0.64     12830
              Symptom       0.00      0.00      0.00      4330
              Vehicle       0.27      0.44      0.33     11184
           VisualWork       0.57      0.58      0.58     29848
          WrittenWork       0.51      0.42      0.46     24218

            micro avg       0.52      0.56      0.54    637273
            macro avg       0.39      0.39      0.38    637273
         weighted avg       0.50      0.56      0.52    637273
```

Figure 14: Precision, recall, and f1 score for each tag and average.

```
, ('B-Station', 392), ('I-Scientist', 391), ('B-AnatomicalStructure', 388), ('B-Vehicle', 377), ('B-Disease', 372), ('B-Food', 362), ('I-SportsManager', 358), ('B-Medication/Vaccine',
355), ('B-SportsManager', 344), ('B-Scientist', 318), ('I-Vehicle', 305), ('B-Cleric', 299), ('I-PublicCorp', 298), ('B-OtherLOC', 291), ('I-Disease', 283), ('B-CarManufacturer', 249),
 ('B-MedicalProcedure', 242), ('B-AerospaceManufacturer', 216), ('B-Drink', 212), ('B-Symptom', 202), ('B-PrivateCorp', 201), ('B-ArtWork', 199), ('B-Clothing', 198), ('I-AerospaceManu
facturer', 178), ('I-PrivateCorp', 170), ('I-MedicalProcedure', 150), ('I-AnatomicalStructure', 148), ('I-Food', 112), ('I-Symptom', 108), ('I-CarManufacturer', 102), ('I-Medication/Va
ccine', 89), ('I-Clothing', 87), ('I-Drink', 84)]
```

Figure 15: Each tag and the number of times they appear in the training data.

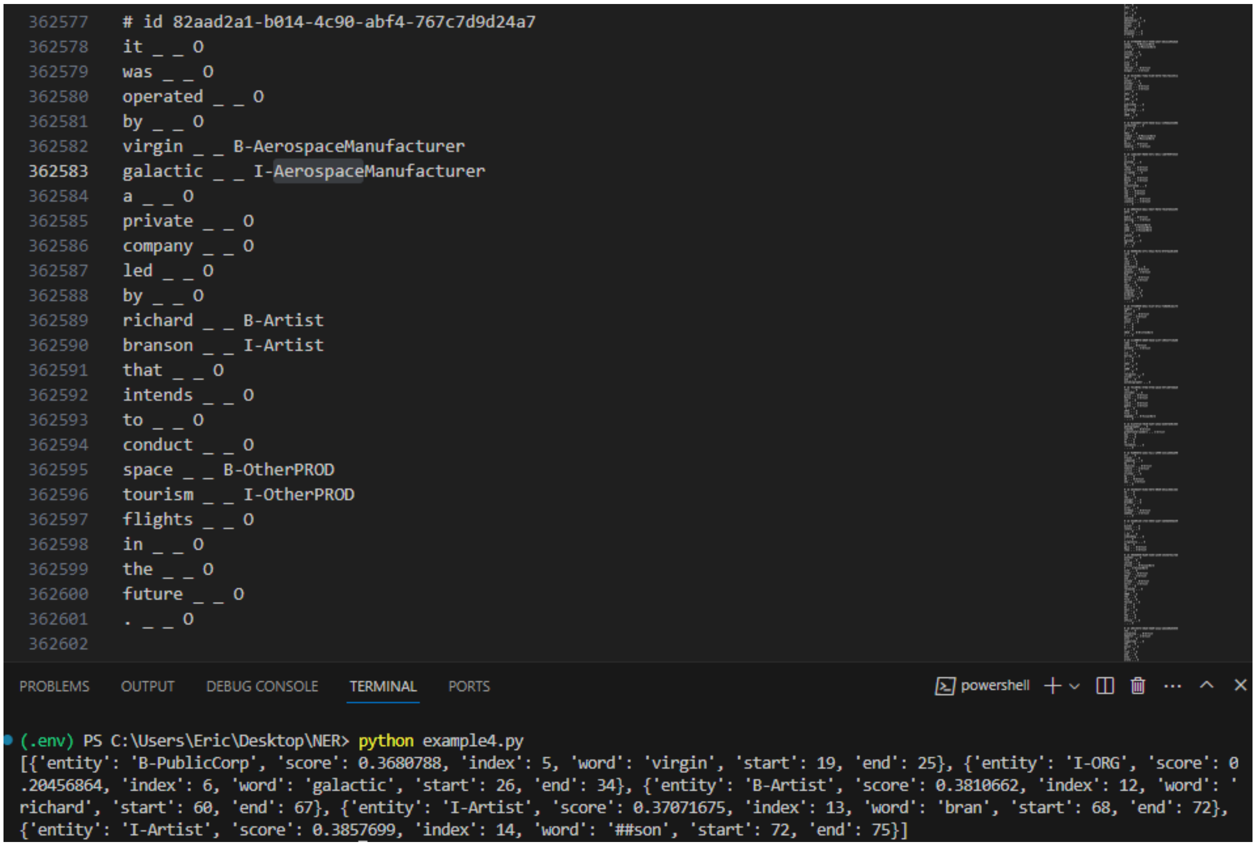


Figure 16: Testing tag prediction on test data entry of tag with low score, AerospaceManufacturer.

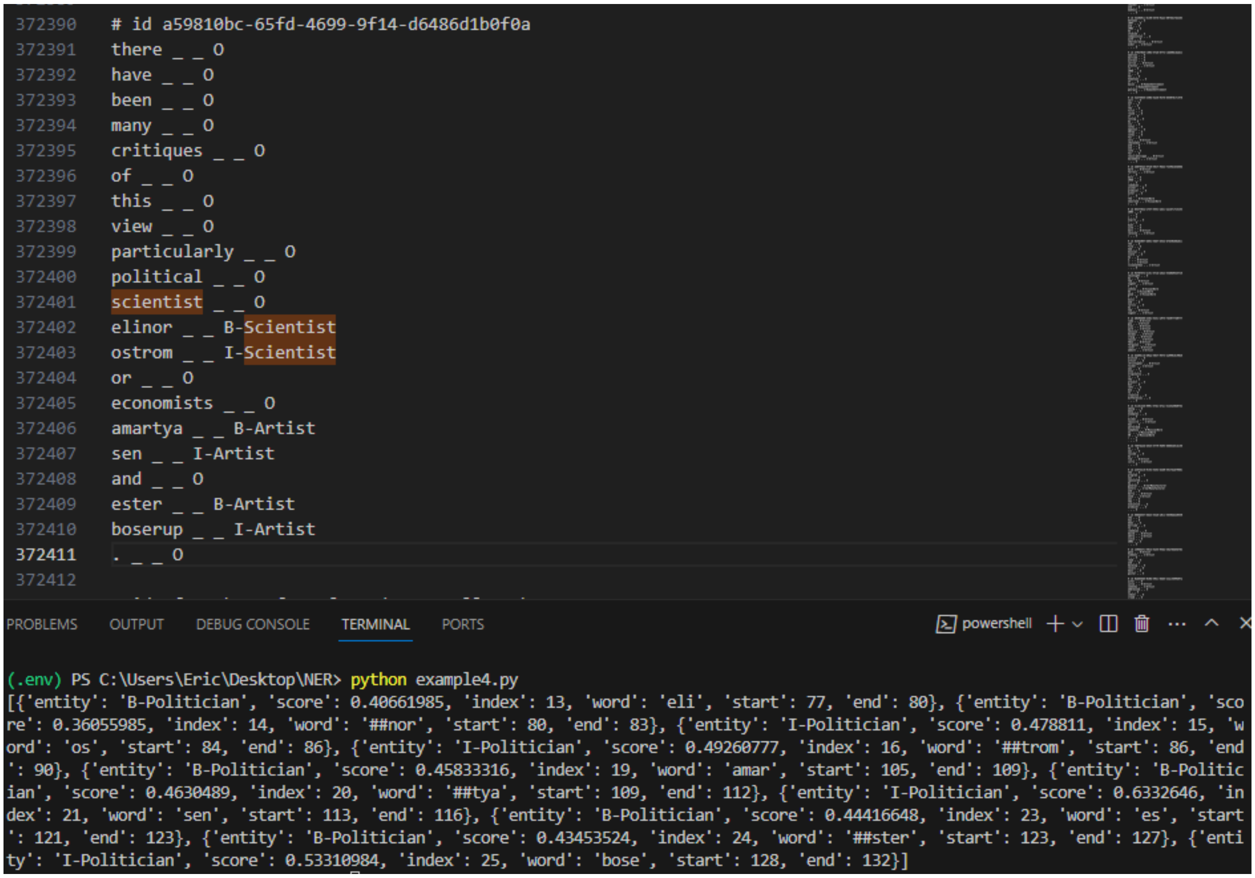


Figure 17: Testing tag prediction on test data entry of tag with zero score, Scientist.

The score achieved in this project was not satisfying, however it did seem like the system was able to identify NEs, just not with the desired prediction. Potential improvements to training this model could be scaling the training data to have more than a maximum 400 occurrences of a unique tag would likely help the score, especially on lower counts. This however, is not likely to help tags which achieved scores of zero. More specific data could help in this case, as many of the entries had mixed tags, such as ‘Scientist’ almost always being included in sentences with ‘Athlete’ or ‘Artist’. Having a cased dataset would also certainly help in identifying NEs as almost all of them would be capitalized.

References

https://multiconer.github.io/paper

https://huggingface.co/docs/transformers/installation

https://huggingface.co/docs/transformers/training

https://github.com/NielsRogge/Transformers-Tutorials/blob/master/BERT/Custom_Named_Entity_Recognition_with_BERT.ipynb