

# Introduction

The objective of this assignment was to create a Natural Language Processing (NLP) model that could extract particular data from sports certificates. The Participant\_name, winning\_position, sports\_name, and the organization\_year are among the targeted data. This document describes the development process's step-by-step methodology, challenges faced, and developments made.

## Architecture of code

For the project, I used modular coding. The following is the structure -

```
NER_SPORTS_TASK/  
|-- config/  
    |-- config.cfg  
|-- data/  
    |-- data.xlsx  
    |-- test_annotations.json  
    |-- train_annotations.json  
|-- env/  
|-- report/  
|-- research/  
    |-- sports_ner.ipynb  
    |-- demo.ipynb  
|-- src/  
    |-- components/  
        |-- data_transformation.py  
        |-- model_training.py  
    |-- constant/  
    |-- entity/  
        |-- artifacts_entity.py  
        |-- config_entity.py  
    |-- exception/
```

```
|-- pipeline/  
    |--train_pipeline.py  
|-- utils/  
    |-- main_utils.py  
|-- app.py  
|-- prediction.py  
|-- requirements.txt  
|-- setup.py
```

## 1. NER\_SPORTS\_TASK

This is the main root folder of the repository.

## 2. config

Inside the config folder the config.cfg file is there. This is a configuration file for training the spacy model. All the necessary parameters are mentioned in this config file.

## 3. data

Inside data folder the main data.xlsx file which is been provided is stored along with the test\_annotations.json and train\_annotations.json files. These json files have been generated in order to train the spacy model.

## 4. report

Inside the report folder the main report for this assignment is mentioned.

## 5. research

Inside this research folder all the necessary experimental jupyter notebooks are stored.

## 6. src

This is the main folder of this assignment. Inside this folder several other folders are mentioned. The components folder include the main components of the assignment i.e. data\_tranformation.py and model\_training.py.

In data transformation stage data is transformed as per the spacy format. In the model\_training stage the spacy model is trained with necessary parameters. The second folder inside the src folder is constant. In this folder all the global variables are mentioned. The third folder inside the src folder is entity. Inside this folder artifacts\_entity and config\_entity are mentioned. Artifacts\_entity includes all the paths of each component. All the artifacts have been stored in the artifacts directory. Config\_entity consists of all the configuration paths needed for each component. The fourth folder in the src folder is an exception. A custom exception has been written to handle errors efficiently. The fifth folder is pipeline. In this folder training pipeline is stored. The sixth folder is utils. In this folder all the necessary utilities are mentioned.

## **Models used**

1. BERT
2. Spacy

This use case is of NER (Named entity recognition). Generally for NER tasks the BERT model works best. I have used a multimodal approach. I have used 2 models to achieve NER. I have used the Spacy and BERT model. The reason I have used two models is because of a lack of data. The task is to extract Participants name, Winning position, Sports name and Year of organization. To extract a person's name is a quite difficult task if you have less data. Hence i have used a pre-trained BERT model for extracting the person name from the certificate descriptions. BERT has a transformers based architecture and requires a huge amount of data to train and perform well. In our scenario the data is very less and we can not fine-tune the BERT model. It will not work well. Hence used a pretrained model to extract participants' names.

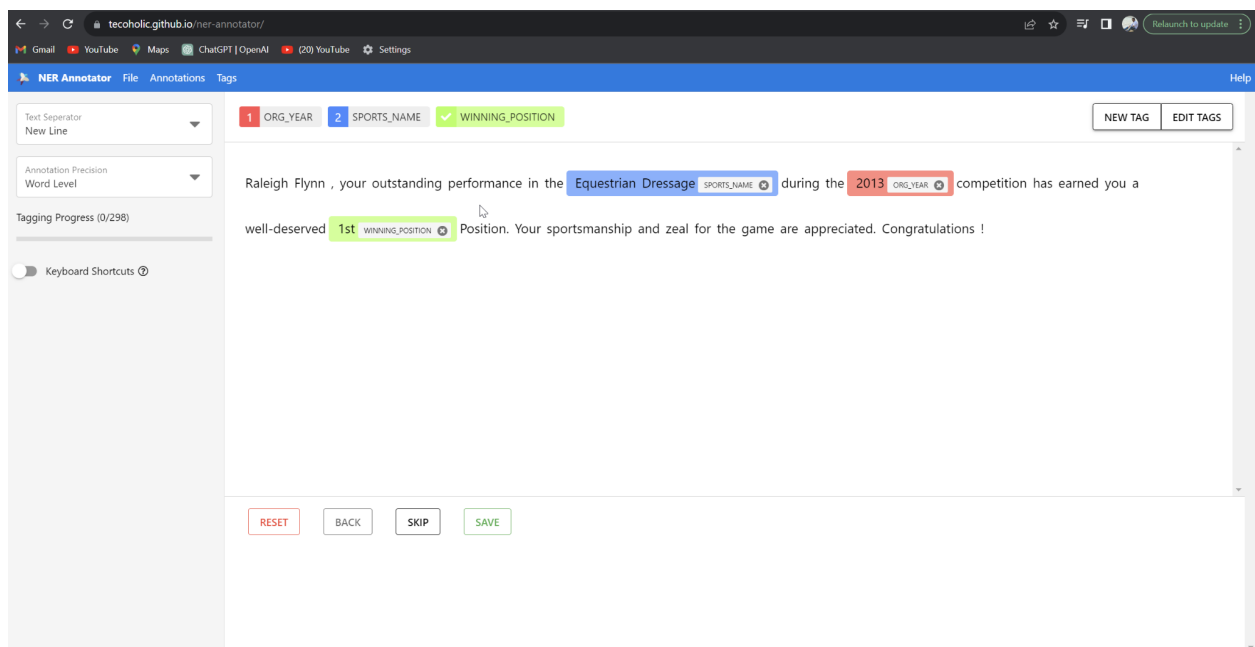
Spacy model has been used to extract Winning position, Sports name and Year of organization. Spacy has a neural network based architecture and can perform well if we have less amount of data. For the training I have used CPU resources. There is no need to train the model on GPU because only 300 rows of data is being given.

## **Challenges faced**

## 1. Labeling:

Custom training has been done for this task to extract Winning position, Sports name and Year of organization. I have used one open source tool to do the POS tagging. The name of the tool is ner-annotator. (github link - <https://github.com/tecoholic/ner-annotator>)

Manually I have done the labeling for the three classes shown in the below image.



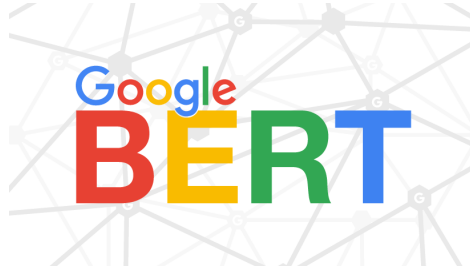
## 2. BERT model interpretability:

It can be difficult to interpret the results of a complicated model like BERT. It took careful examination of the tokenized sequences to extract the pertinent data from the predicted labels.

# Tools used

## 1. Transformers

Hugging Face's transformers library provided important tools for working with pre-trained models, including BERT



## 2. Spacy

Spacy has been used to extract Winning position, Sports name and Year of organization. It made information extraction from the new Certificate description easier.



## 3. PyTorch

We used pytorch as a deep learning framework.



## 4. Pandas

Pandas is used for data manipulation and for reading and writing the data.



## 5. Fast api

Fast api is used to create an UI. I have created 1 endpoint for training.



## **Methodology**

I have followed the below methodology

### **1. Data Collection:**

The data has been provided in the form of an excel sheet. There were two sheets in the main file. One is 'Completed' and the other is 'Data-Pending'. There were 300 certificates data presented in the completed sheet for training. Each data sample (certificate description) has information about Participant name, Winning position, sports name and organization year. Then we arranged the dataset in following columns in a DataFrame format:

'Certificate description', 'Participants name', "Sport name", 'Position', and 'Organization year'.

### **2. Data preprocessing:**

We processed the data by splitting the data into training data and test data. The data has been splitted in the ratio of 90:10. Roughly 270 samples for training and 30 samples for testing.

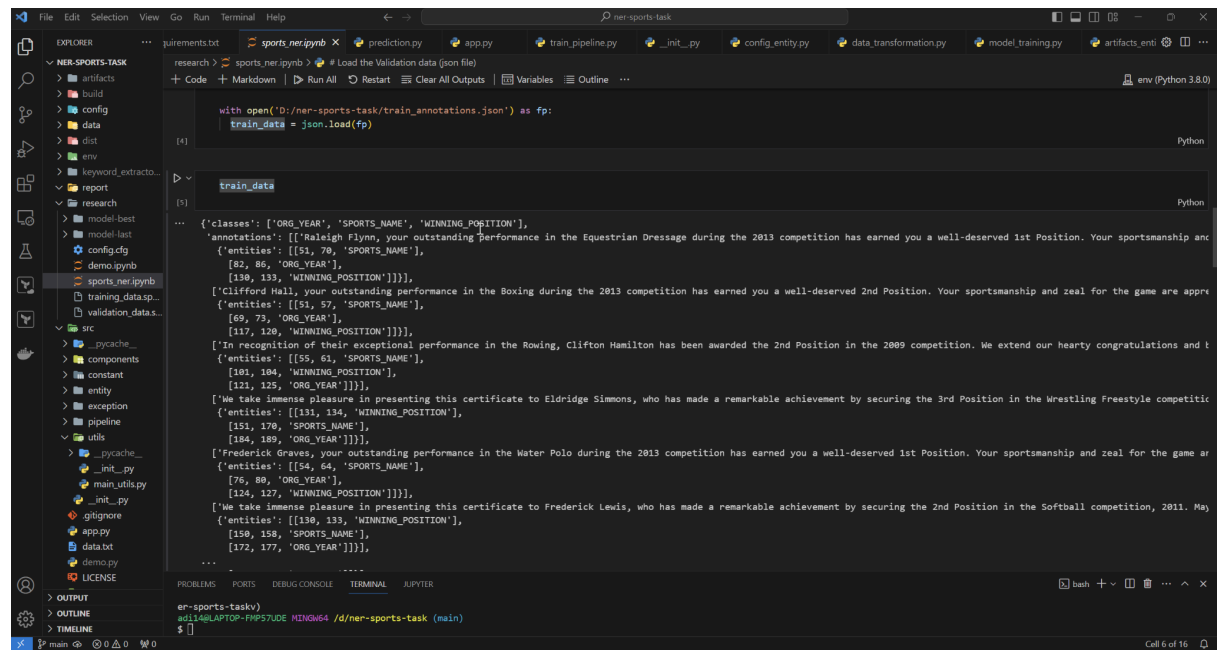
- **For BERT -**

We used a pre-trained BERT model. The model '`dsllim/bert-base-NER`' has been trained on four types of entities: location (LOC), organizations (ORG), person (PER) and Miscellaneous (MISC). Specifically, this model is a bert-base-cased model that was fine-tuned on the English version of the standard CoNLL-2003 Named Entity Recognition dataset. I have used this model to only extract the person's name from the certificate

descriptions. I have used the AutoTokenizer class and used the pre-trained tokenizer.

- **For Spacy -**

I have used the open source tool for annotations. When we export the annotations it provides a json file with the annotations and classes.



The screenshot shows a VS Code editor with a project named 'ner-sports-task'. The file explorer on the left shows a directory structure with files like 'train\_data.json', 'validation\_data.json', and 'ner.py'. The main editor window shows a Python script with the following code:

```
with open('D:/ner-sports-task/train_annotations.json') as fp:
    train_data = json.load(fp)
```

The output of the script is displayed in the console, showing a list of JSON records. Each record contains a list of entities (e.g., 'ORG\_YEAR', 'SPORTS\_NAME', 'WINNING\_POSITION') and their corresponding values (e.g., '1913', 'Equestrian Dressage', '1st Position').

Then I have converted these json records in the spacy required format (.spacy) using the make\_doc() method.

### 3. Model selection:

For Named Entity Recognition (NER), I have selected BERT (Bidirectional Encoder Representations from Transformers) as the important model. BERT is known for its better NLP task performance. I have extracted the participants' names using a BERT pretrained model. I haven't fine-tuned the BERT model due to lack of data.

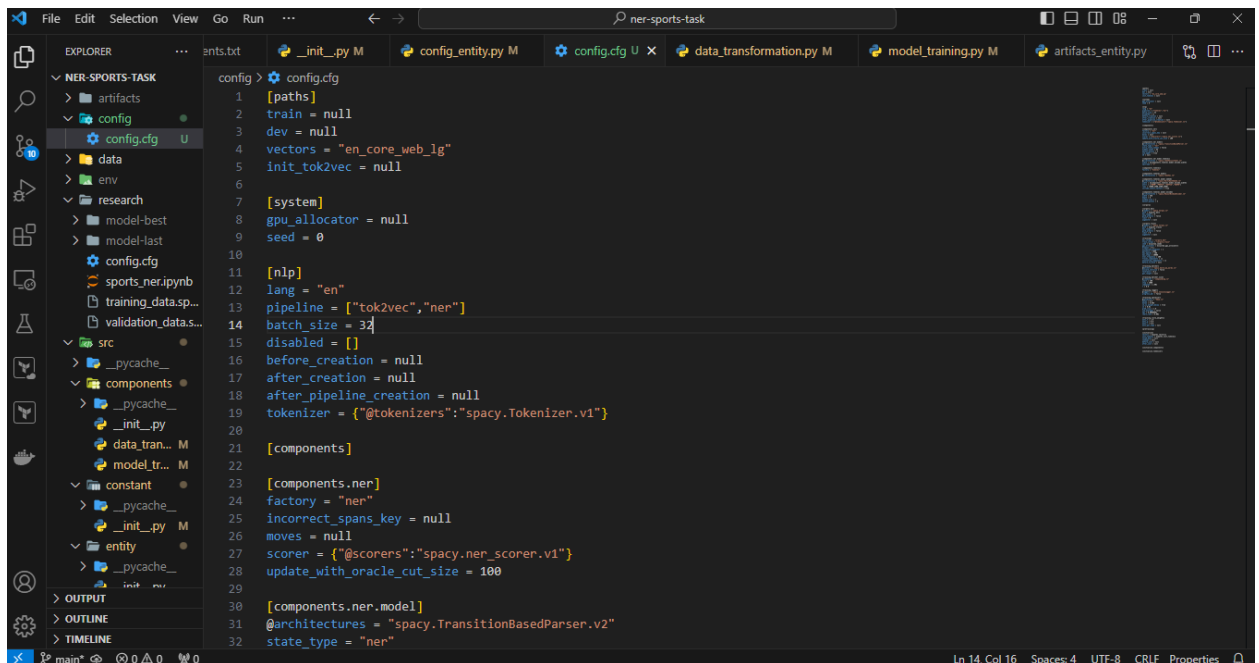
I have used the Spacy model to extract the other information. I observed that the training data has the same pattern hence it will be an easy task to use a spacy model to learn on the small amount of data.

### 4. Training the model:

The spacy model was trained on the dataset with a config file. I have downloaded the config file with following command -

```
!python -m spacy init config config.cfg --lang en
--pipeline ner --optimize accuracy
```

Here the language is 'english', pipeline is set to 'ner' since this is a ner task and the optimization is based on the accuracy. After downloading the config file I have done a little hyperparameter tuning. I have changed the epochs and batch size as per our data.



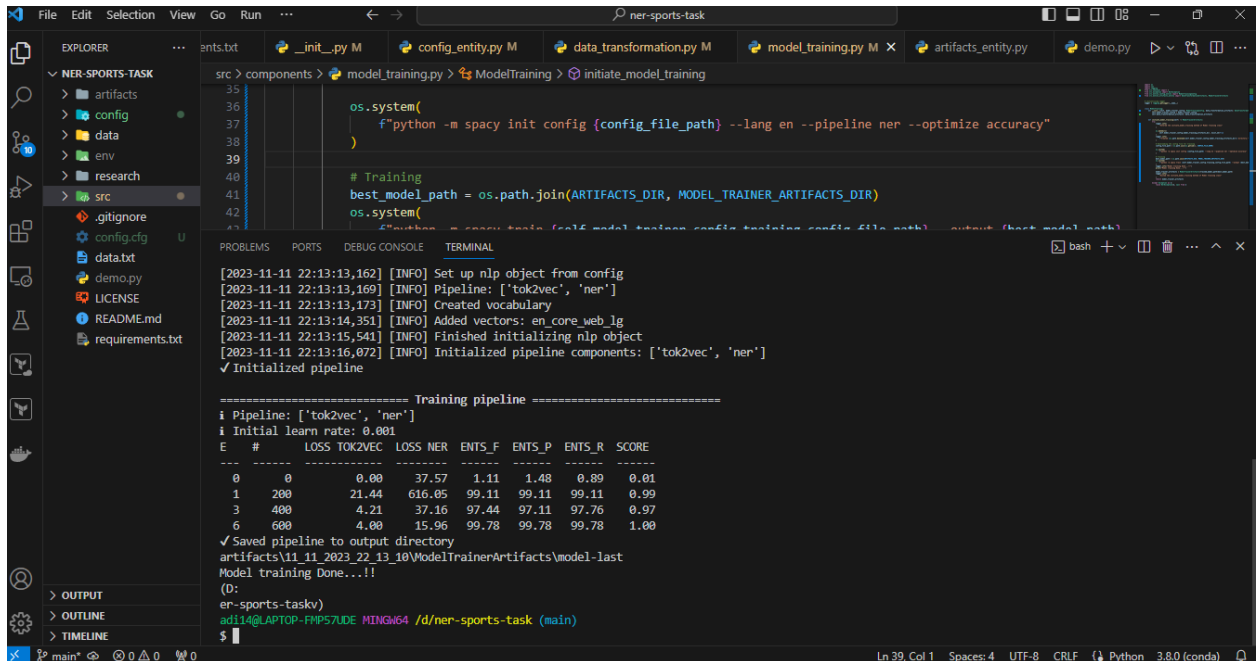
```
config > config.cfg
1  [paths]
2  train = null
3  dev = null
4  vectors = "en_core_web_lg"
5  init_tok2vec = null
6
7  [system]
8  gpu_allocator = null
9  seed = 0
10
11 [nlp]
12 lang = "en"
13 pipeline = ["tok2vec", "ner"]
14 batch_size = 32
15 disabled = []
16 before_creation = null
17 after_creation = null
18 after_pipeline_creation = null
19 tokenizer = {"@tokenizers": "spacy.Tokenizer.v1"}
20
21 [components]
22
23 [components.ner]
24 factory = "ner"
25 incorrect_spans_key = null
26 moves = null
27 scorer = {"@scorers": "spacy.ner_scorer.v1"}
28 update_with_oracle_cut_size = 100
29
30 [components.ner.model]
31 @architectures = "spacy.TransitionBasedParser.v2"
32 state_type = "ner"
```

This is the glimpse of the config file. 'Adam.V1' optimizer has been used to optimize the loss. 'En\_core\_web\_lg' english pre-trained model is used for fine-tuning.

## 5. Performance metrics:

The performance of the spacy model has been checked on the F1 score, Precision and recall.





```
src > components > model_training.py > ModelTraining > initiate_model_training
35
36     os.system(
37         f"python -m spacy init config {config_file_path} --lang en --pipeline ner --optimize accuracy"
38     )
39
40     # Training
41     best_model_path = os.path.join(ARTIFACTS_DIR, MODEL_TRAINER_ARTIFACTS_DIR)
42     os.system(
43         f"python -m spacy train {self.model_training_config_file_path} --output {best_model_path}"
44     )
```

```
[2023-11-11 22:13:13,162] [INFO] Set up nlp object from config
[2023-11-11 22:13:13,169] [INFO] Pipeline: ['tok2vec', 'ner']
[2023-11-11 22:13:13,173] [INFO] Created vocabulary
[2023-11-11 22:13:14,351] [INFO] Added vectors: en_core_web_lg
[2023-11-11 22:13:15,541] [INFO] Finished initializing nlp object
[2023-11-11 22:13:16,072] [INFO] Initialized pipeline components: ['tok2vec', 'ner']
✓ Initialized pipeline

===== Training pipeline =====
i Pipeline: ['tok2vec', 'ner']
i Initial learn rate: 0.001
E  #      LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
-----
0   0          0.00      37.57    1.11    1.48    0.89    0.01
1  200       21.44     616.05   99.11   99.11   99.11   0.99
3  400        4.21      37.16   97.44   97.11   97.76   0.97
6  600        4.00      15.96   99.78   99.78   99.78   1.00
✓ Saved pipeline to output directory
artifacts\11_11_2023_22_13_10\ModelTrainerArtifacts\model-last
Model training Done...!!
(D:
er-sports-taskv)
adi14@LAPTOP-FMP57UDE MINGW64 /d/ner-sports-task (main)
$
```

In the diagram above you can see E, #, LOSS TOK2VEC, LOSS NER, ENTS\_F, ENTS\_P, ENTS\_R, SCORE.

- The letter ‘E’ represents the number of epochs. By default when you download the config file the epoch is set to 0 means it will optimize as much as possible. So I changed the epoch size to 20 as per the model accuracy.
- The ‘#’ means the iterations and batches. By default it comes with the batch size of 1000. I have changed the batch size to 16.
- Tok2vec is a model used to learn how to produce suitable vectors for tokens. The overall loss for this model has been given here.
- The overall loss has been calculated under the **LOSS\_NER** column
- The **ENTS\_F** represents the F1-score at each epoch.
- The **ENTS\_P** represents the Precision at each epoch.
- The **ENTS\_R** represents the Recall at each epoch.
- The **SCORE** represents an overall model score.

As you can see in the diagram the loss has been gradually decreasing and the overall score is gradually increasing.