**Named Entity Recognition**

About the Dataset

File ner\_dataset.csv maps sentence number (Sentence #) to first word (Word) of respective sentence. Then each word is mapped to its respective POS tag (POS) and NER tag (Tag). Ignore the POS tag column for scope of this case-study. For every sentence just consider the word and its NER tag.

Sample mapping of a sentence:

*Today O*

*Micheal B-PER*

*Jackson I-PER*

*and O*

*Mark B-PER*

*ate O*

*lasagna O*

*at O*

*New B-geo*

*Delhi I-geo*

*. O*

**Sequence tagging scheme**: IOB2

* I : inside : word is inside a chunk
* O : outside : word belongs to no chunk
* B : beginning : word is the beginning of a chunk

**Columns**:

* **Sentences #** : sentence number
* **Word** : word to be classified
* **POS** : POS tags for respective word
* **Tag** : NER tags for respective word

**Probable tasks (Below pointers are for direction purpose only):**

* divide the dataset into 3 parts:
  + train
  + validation
  + test (at least 20%)

Done

* Identify the matrices for evaluating model's performance.

Weighted F1-score

* Pre-process the data such that words of each sentence is mapped to their respective NER tags. Done
* Develop a baseline model which takes a sentence (list of words) as input and predicts NER tag for each word in that sentence. Spacy NER Model
* Identify the short comings of the baseline model.

1. Higher convergence time – More than 1 hours for 1 Mn records without GPU.
2. Resource Intensive: Training custom NER models in spaCy can be resource-intensive, especially when dealing with large datasets or complex model architectures. This can pose challenges for users with limited computational resources
3. Pretrained Models May Not Be Customizable: While spaCy provides pretrained models for NER, these models may not always be easily customizable to suit specific domains or use cases. Fine-tuning or customizing pretrained models can be challenging compared to other frameworks like TensorFlow or PyTorch.
4. Limited Control Over Model Architecture: While spaCy provides an easy-to-use interface for NER, it may not offer as much flexibility and control over the underlying model architecture compared to lower-level libraries like TensorFlow or PyTorch. This can be a limitation for researchers or developers who require fine-grained control over model components.

* Develop a new model which overcome the shortcomings of baseline model.

LSTM Model

1. **Model Architecture**: The model uses a simple LSTM layer. While LSTMs are powerful, they might not be the best choice for NER tasks. More advanced architectures like Bi-directional LSTMs or Transformer-based models (like BERT) could potentially yield better results.
2. **Embedding Layer**: The model uses a basic embedding layer to convert words into vectors. Pre-trained word embeddings like Word2Vec, GloVe, or even contextual embeddings from models like BERT or ELMo could improve the performance by leveraging pre-existing linguistic knowledge.
3. **Hyperparameter Tuning**: The model’s hyperparameters (like the learning rate, batch size, number of LSTM units, etc.) are set manually. Using a more systematic approach like grid search or random search could help find a better set of hyperparameters.
4. **Evaluation Metric**: The model uses weighted F1 score as the evaluation metric. While F1 score is a good metric for imbalanced datasets, it might be beneficial to also look at other metrics like precision, recall, and the F1 score for individual classes (not just the weighted average) to get a more comprehensive understanding of the model’s performance. However, you might want to consider using more detailed metrics like precision, recall, and F1-score for each tag type. This can give you a better understanding of how your model is performing for each entity type.
5. **Error Analysis**: The code does not include any error analysis. Looking at the types of errors the model is making can provide insights on how to improve it. For example, if certain types of entities are often misclassified, additional features could be engineered to help the model distinguish them.
6. **Data Preprocessing**: The code could be optimized to preprocess data more efficiently. For example, the preprocess\_data function uses a for loop to iterate over the DataFrame, which is not the most efficient method in pandas.

* Identify future scope to further optimise the model.

1. **Class Imbalance**: NER datasets often have a class imbalance problem, where the ‘O’ (Outside) tag is far more common than other tags. This can cause the model to be biased towards predicting ‘O’. Techniques like oversampling the minority class, undersampling the majority class, or using class weights can help mitigate this issue.
2. Domain-Specific Entity Recognition: General-purpose NER models may struggle to identify entities in domain-specific texts accurately1. For instance, if you’re working in the medical domain, you may need to recognize medical terms, drug names, and specific medical conditions1

**System design tasks (Below pointers are for direction purpose only):**• Design system architecture to deploy ML Model in production  
• How do you perform canary build?  
• What should be the strategy for ML Model Monitoring?  
• How do you perform load and stress testing?  
• How do you track, monitor and audit ML training?  
• Design framework for continuous delivery and automation of machine learning tasks.

**Deliverables:**  
- Jupyter notebook (or equivalent) showcasing your work  
- Powerpoint presentation clearly explaining the approach and findings.  
- System design architecture (if applicable) and explanations.