Ruby Prajapati (231110042)

Question Number : 5

Comparison of chatGPT , indicBERT and indicNER :

From the comparison of precision, recall, and macro F1 scores between the ChatGPT, indicBERT, and indicNER models against the ground truth labels, several observations can be made:

- **ChatGPT Performance:**
  ChatGPT achieves a relatively high precision and recall for the 'O' class (non-named entities), indicating its effectiveness in identifying tokens that are not named entities. However, it performs poorly for most named entity classes, with low precision and recall values. This suggests that while ChatGPT can recognize non-entity tokens well, it struggles with identifying specific named entities.
  The macro F1 score for ChatGPT is 0.32, indicating an overall moderate performance across all named entity classes. However, this score is mainly driven by the high precision and recall for the 'O' class, while the F1 scores for other classes are significantly lower.

  OUTPUT :

	precision	recall	f1-score	support
B-LOC	0.20	0.17	0.18	6
B-MISC	0.00	0.00	0.00	31
B-ORG	0.20	0.50	0.29	2
B-PER	0.50	0.50	0.50	6
I-LOC	0.00	0.00	0.00	2
I-MISC	0.00	0.00	0.00	18
I-ORG	1.00	0.17	0.29	6
I-PER	0.67	0.67	0.67	6
O	0.91	0.99	0.95	519
accuracy			0.88	596
macro avg	0.39	0.33	0.32	596
weighted avg	0.82	0.88	0.84	596

```
Macro F1 Score: 0.31847877596491275
```

- **indicBERT Performance:**
  indicBERT also performs poorly in identifying named entities, with low precision, recall, and macro F1 scores across all classes. Like ChatGPT, indicBERT achieves relatively high precision and recall for the 'O' class but struggles with named entity recognition. The macro F1 score for indicBERT is even lower than that of ChatGPT, at 0.14. This indicates that indicBERT's performance is consistently poor across all named entity classes, leading to a lower overall F1 score.

  OUTPUT :

	precision	recall	f1-score	support
I-MISC	0.00	0.00	0.00	18
I-ORG	0.00	0.00	0.00	6
I-PER	0.20	0.17	0.18	6
B-MISC	0.00	0.00	0.00	31
I-LOC	0.00	0.00	0.00	2
O	0.89	0.91	0.90	519
B-PER	0.14	0.17	0.15	6
B-ORG	0.00	0.00	0.00	2
B-LOC	0.00	0.00	0.00	6
micro avg	0.84	0.80	0.82	596
macro avg	0.14	0.14	0.14	596
weighted avg	0.78	0.80	0.79	596

```
Macro F1 Score: 0.1370951703967879
```

- **indicNER Performance:**
  indicNER shows a slightly better performance compared to both ChatGPT and indicBERT. It achieves higher precision, recall, and macro F1 scores for most named entity classes, although the scores are still relatively low.
  The macro F1 score for indicNER is 0.26, indicating a moderate overall performance. While indicNER performs better than indicBERT, it still struggles with named entity recognition, particularly for classes with low support.

  OUTPUT :

	precision	recall	f1-score	support
I-MISC	0.00	0.00	0.00	18
I-ORG	0.50	0.33	0.40	6
I-PER	0.20	0.17	0.18	6
B-MISC	0.00	0.00	0.00	31
I-LOC	0.00	0.00	0.00	2
O	0.91	0.93	0.92	519
B-PER	0.22	0.33	0.27	6
B-ORG	0.12	0.50	0.20	2
B-LOC	0.25	0.50	0.33	6
micro avg	0.86	0.82	0.84	596
macro avg	0.25	0.31	0.26	596
weighted avg	0.80	0.82	0.81	596

```
Macro F1 Score: 0.255528568123988
```

- **Overall Observations:**
  All models struggle with identifying named entities, particularly for classes with low support such as B-MISC, I-LOC, I-MISC, and B-ORG.
  The 'O' class (non-named entities) is consistently identified with high precision and recall by all models, indicating their effectiveness in recognizing tokens that are not named entities.

Parameters :

- **num_train_epochs:** This hyperparameter specifies the number of times the entire training dataset is passed through the model during training. Increasing the number of epochs can allow the model to see more examples and potentially learn better representations, but too many epochs can lead to overfitting.

- **learning_rate:** This hyperparameter determines the step size at which the model weights are updated during training. A higher learning rate can lead to faster convergence but may also cause instability or overshooting of optimal weights. Conversely, a lower learning rate may lead to slower convergence but can help the model find more precise weights.

- **per_device_train_batch_size:** This hyperparameter sets the number of training examples processed simultaneously on each device during training. A larger batch size can lead to faster training but may require more memory and could result in a decrease in generalization performance. Conversely, a smaller batch size may lead to slower training but could result in better generalization.

- **weight_decay:** This hyperparameter controls the amount of regularization applied to the model weights during training. Regularization helps prevent overfitting by penalizing large weight values. Increasing the weight decay value increases the amount of regularization applied, which can help improve generalization performance.

- **gradient_accumulation_steps:** This hyperparameter specifies the number of gradient accumulation steps before updating the model weights. It allows for effectively increasing the batch size without increasing memory usage, which can help stabilize training and improve convergence.

- **warmup_steps:** This hyperparameter determines the number of initial steps during which the learning rate is gradually increased from zero to its initial value. Warm-up steps help stabilize training and prevent the model from getting stuck in suboptimal solutions.

- **eval_steps:** This hyperparameter specifies the frequency at which the model is evaluated on the validation dataset during training. Increasing the evaluation frequency can provide more frequent feedback on the model's performance but may also increase training time.

- **evaluation_strategy:** This hyperparameter determines when to evaluate the model on the validation dataset. Options include "epoch" (evaluate at the end of each epoch) or "steps" (evaluate after a certain number of steps). Choosing the appropriate evaluation strategy depends on the dataset and training objectives.

Hyper parameter Values used :

- **num_train_epochs = 2:** Three epochs are chosen to allow the model to see the training data multiple times, enabling it to learn the patterns effectively without overfitting. This value strikes a balance between underfitting and overfitting.

- **learning_rate = 3e-5:** This learning rate is commonly used for fine-tuning pre-trained language models like BERT. It provides a moderate step size for updating the model weights during training, ensuring a stable learning process without convergence issues.

- **per_device_train_batch_size = 8:** A batch size of 8 is chosen to balance between training efficiency and memory constraints. It allows for processing a reasonable number of examples simultaneously on each GPU, optimizing training speed while avoiding out-of-memory errors.

- **weight_decay = 0.01:** A weight decay of 0.01 applies moderate regularization to the model weights during training, helping prevent overfitting by penalizing large weight values. This value is commonly used and has been found to work well in practice.

- **gradient_accumulation_steps = 1:** With a gradient accumulation step of 1, gradients are accumulated and model weights are updated after processing each batch. This helps maintain stability during training and ensures that the model converges smoothly.

- **warmup_steps = 100:** Warmup steps gradually increase the learning rate from zero to its initial value over a specified number of steps. This helps stabilize training and prevents the model from getting stuck in suboptimal solutions, especially at the beginning of training.

- **eval_steps = 1000:** The model is evaluated on the validation dataset every 1000 steps during training. This frequency provides regular feedback on the model's performance without significantly increasing training time or overhead.

- **evaluation_strategy = "epoch":** The model is evaluated at the end of each epoch. This strategy allows for comprehensive evaluation of the model's performance after processing the entire training dataset, helping monitor progress and detect overfitting.

indicBERT Model :

Hyper parameters:

```
# # trainer.args
trainer.args.num_train_epochs = 2
trainer.args.learning_rate = 3e-5
trainer.args.per_device_train_batch_size = 16
trainer.args.weight_decay = 0.01
trainer.args.gradient_accumulation_steps = 2
trainer.args.warmup_steps = 500
trainer.args.eval_steps = 1000
trainer.args.evaluation_strategy = "epoch"
```

Output :

```
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will ins
  warnings.warn('Was asked to gather along dimension 0, but all '
[3062/3062 2:29:54, Epoch 1/2]
```

ch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
0	0.273200	0.260686	0.710377	0.709194	0.709785	10213	0.588476	0.513489	0.548431	9786	0.726011	0.688020	0.706505	10568	0.679617	0.639219	0.658799	0.919289
1	0.232000	0.240391	0.727057	0.751689	0.739168	10213	0.595166	0.561108	0.577635	9786	0.739307	0.724546	0.731852	10568	0.690896	0.681290	0.686060	0.925185

Eval Matrix :

```
***** eval metrics *****
  epoch                   =        3.0
  eval_LOC_f1             =     0.7093
  eval_LOC_number         =      10213
  eval_LOC_precision      =     0.7009
  eval_LOC_recall         =     0.7179
  eval_ORG_f1             =     0.5503
  eval_ORG_number         =       9786
  eval_ORG_precision      =     0.5676
  eval_ORG_recall         =     0.5339
  eval_PER_f1             =     0.6931
  eval_PER_number         =      10568
  eval_PER_precision      =     0.7016
  eval_PER_recall         =     0.6847
  eval_loss               =      0.272
  eval_overall_accuracy   =     0.9171
  eval_overall_f1         =     0.6538
  eval_overall_precision  =     0.6602
  eval_overall_recall     =     0.6475
  eval_runtime            = 0:04:19.98
  eval_samples_per_second =     51.773
  eval_steps_per_second   =      3.239
```

Train matrix :

```
***** train metrics *****
  eval_LOC_f1             =     0.7642
  eval_LOC_number         =      14841
  eval_LOC_precision      =     0.7544
  eval_LOC_recall         =     0.7741
  eval_ORG_f1             =     0.6246
  eval_ORG_number         =      14082
  eval_ORG_precision      =     0.6468
  eval_ORG_recall         =     0.6039
  eval_PER_f1             =     0.7789
  eval_PER_number         =      15614
  eval_PER_precision      =     0.7882
  eval_PER_recall         =     0.7698
  eval_loss               =      0.205
  eval_overall_accuracy   =     0.9386
  eval_overall_f1         =     0.7262
  eval_overall_precision  =     0.7338
  eval_overall_recall     =     0.7188
  eval_runtime            = 0:06:26.00
  eval_samples_per_second =     51.813
  eval_steps_per_second   =      3.238
```

indic NER Model :

Parameters :

```
# # trainer.args
trainer.args.num_train_epochs = 2
trainer.args.learning_rate = 3e-5
trainer.args.per_device_train_batch_size = 16
trainer.args.weight_decay = 0.01
trainer.args.gradient_accumulation_steps = 2
trainer.args.warmup_steps = 500
trainer.args.eval_steps = 1000
trainer.args.evaluation_strategy = "epoch"
```

Output :

[1406/1406 1:24:53, Epoch 1/2]

ch	Training Loss	Validation Loss	Loc Precision	Loc Recall	Loc F1	Loc Number	Org Precision	Org Recall	Org F1	Org Number	Per Precision	Per Recall	Per F1	Per Number	Overall Precision	Overall Recall	Overall F1	Overall Accuracy
0	1.348600	0.166736	0.815631	0.859395	0.836941	10213	0.668689	0.698140	0.683098	9786	0.805899	0.837623	0.821455	10568	0.765315	0.800242	0.782389	0.947999
1	0.138600	0.169804	0.813226	0.857339	0.834700	10213	0.689109	0.689250	0.689180	9786	0.804348	0.836771	0.820239	10568	0.771625	0.796414	0.783824	0.948078

Eval matrix :

```
***** eval metrics *****
  epoch                   =        3.0
  eval_LOC_f1             =     0.8358
  eval_LOC_number         =      10213
  eval_LOC_precision      =      0.816
  eval_LOC_recall         =     0.8565
  eval_ORG_f1             =     0.6829
  eval_ORG_number         =       9786
  eval_ORG_precision      =     0.6761
  eval_ORG_recall         =     0.6898
  eval_PER_f1             =     0.8174
  eval_PER_number         =      10568
  eval_PER_precision      =     0.8045
  eval_PER_recall         =     0.8307
  eval_loss               =     0.1978
  eval_overall_accuracy   =     0.9467
  eval_overall_f1         =     0.7808
  eval_overall_precision  =     0.7678
  eval_overall_recall     =     0.7942
  eval_runtime            = 0:04:59.06
  eval_samples_per_second =     45.007
  eval_steps_per_second   =      2.815
```

Train matrix :

```
***** train metrics *****
  eval_LOC_f1             =     0.9141
  eval_LOC_number         =      14841
  eval_LOC_precision      =     0.8974
  eval_LOC_recall         =     0.9313
  eval_ORG_f1             =     0.8459
  eval_ORG_number         =      14082
  eval_ORG_precision      =     0.8436
  eval_ORG_recall         =     0.8482
  eval_PER_f1             =     0.9101
  eval_PER_number         =      15614
  eval_PER_precision      =     0.9046
  eval_PER_recall         =     0.9157
  eval_loss               =     0.0615
  eval_overall_accuracy   =     0.9798
  eval_overall_f1         =     0.8913
  eval_overall_precision  =     0.8831
  eval_overall_recall     =     0.8996
  eval_runtime            = 0:07:22.69
  eval_samples_per_second =     45.178
  eval_steps_per_second   =      2.824
```

Comparison of outputs of both models :

Based on the provided output of the named entity recognition (NER) models, here's a comparison report:

- **Training and Validation Loss:**

  Both models show a decrease in training loss from epoch 0 to epoch 1, indicating improved model fit over epochs. However, the validation loss slightly increases from epoch 0 to epoch 1 for both models, suggesting potential overfitting.

- **Precision, Recall, and F1-scores:**

  For the 'Loc' (location) entity class, both models exhibit similar precision, recall, and F1-scores across epochs, with indicNER showing slightly higher values compared to indicBERT.
  For the 'Org' (organization) entity class, indicNER outperforms indicBERT in terms of precision, recall, and F1-scores, showing an improvement from epoch 0 to epoch 1.
  Similarly, for the 'Per' (person) entity class, indicNER shows higher precision, recall, and F1-scores compared to indicBERT across epochs.
  Overall, indicNER demonstrates better performance in identifying named entities compared to indicBERT, as indicated by higher precision, recall, and F1-scores for all entity classes.

- **Overall Accuracy:**
  Both models achieve high overall accuracy, with indicNER showing a slightly higher accuracy compared to indicBERT. This suggests that indicNER performs better in accurately identifying named entities overall.

Based on the provided metrics and comparison between indicNER and indicBERT models for named entity recognition (NER), it appears that indicNER is working well compared to indicBERT.

The reasons for this are as follows:

- **Higher Precision, Recall, and F1-Scores:** indicNER consistently exhibits higher precision, recall, and F1-scores across all entity classes (Loc, Org, Per) compared to indicBERT. This indicates that indicNER is more effective in accurately identifying named entities in the text.

- **Overall Accuracy:** indicNER achieves a slightly higher overall accuracy compared to indicBERT, indicating that it performs better in accurately classifying tokens into named entity categories.

- **Consistent Improvement:** indicNER shows improvements in precision, recall, and F1-scores from epoch 0 to epoch 1, suggesting that the model is learning and adapting well over epochs. On the other hand, indicBERT's performance remains relatively stable across epochs.

- **Better Handling of Entity Classes:** indicNER performs particularly well in identifying entities such as 'Org' (organization) and 'Per' (person), where it achieves higher precision, recall, and F1-scores compared to indicBERT. This indicates that indicNER is better equipped to handle a diverse range of named entity classes.

Overall, indicNER is working well compared to indicBERT due to its consistently higher precision, recall, and F1-scores, better overall accuracy, and improved handling of different named entity classes.