Paper PP07

Bio BERT-Powered Event Classification - A Machine Learning Approach in Python

Gayathiri Premkumar, SENSAN Biosciences, Chennai, India
Premkumar, Rajalakshmi Engg. College, Chennai, India
Raja lingam, CTS, Chennai, India

ABSTRACT

The exponential growth of unstructured biomedical text data presents a significant challenge for researchers seeking to extract critical information, such as specific events and relationships. This paper details a **machine learning approach implemented in Python for advanced biomedical event classification**, leveraging the contextual understanding capabilities of **Bio BERT.** We demonstrate how this Bio BERT-powered framework enables accurate and efficient extraction of complex biomedical events, thereby facilitating tasks such as **pharmacovigilance, disease progression analysis, Clinical Researchers, Healthcare Professionals & biomarker discovery, and ultimately accelerating knowledge discovery in the life sciences.**

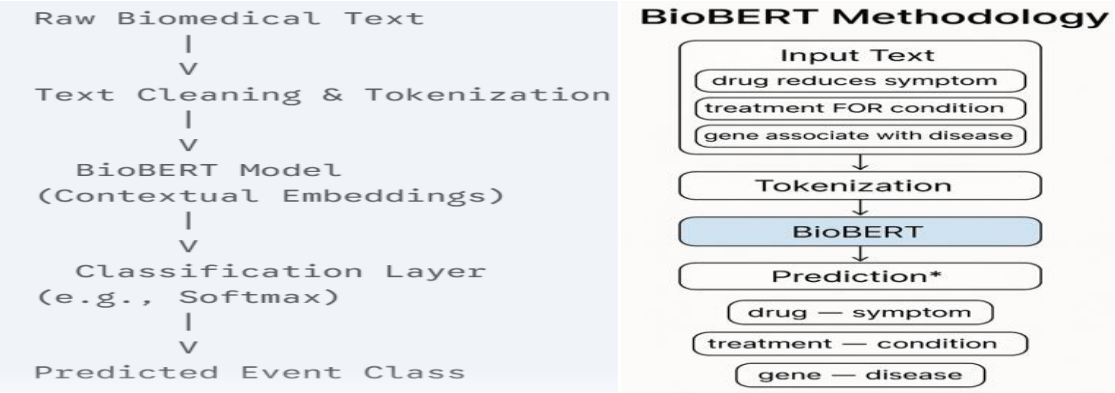
INTRODUCTION

Our methodology outlines the pipeline **from raw text input and entity recognition (where relevant) to event type prediction,** focusing on the fine-tuning of the Bio BERT model for domain-specific event identification. By harnessing Bio Bert's pre-trained knowledge from vast biomedical corpora, the approach aims to overcome the limitations of traditional rule-based or feature-engineered methods. This (Bio BERT) Bidirectional Encoder Representations from Transformers defines Domain-specific biomedical language representation model built on top of BERT. We have to pretrain this model by using Medical Dictionary, Research papers.

Data Flow Diagram & Methodology

We can understand the data flow & how this methodology works by seeing the snippet below.

Flowcharts: Illustrating the sequence of steps from raw text to final classification.
Block Diagrams: Showing modules (e.g., "Tokenizer", "Bio BERT Encoder", "Classifier") with arrows indicating data flow.
Data Representation Snapshots: Small examples of text, tokens, embeddings (as numerical vectors), and final class labels.



Flow Chart & BIO BERT Methodology

Environmental Configuration and Model Loading

For this unstructured text classification, we are using Python code in Machine language model. Before starting our model implementation, we must install necessary pretrained models in python environment. This snippet sets up the tools needed to build a BERT-based text classification model using Py Torch and Hugging Face Transformers. It's commonly used in NLP tasks like sentiment analysis, entity recognition, or biomedical text classification (like Bio BERT). Those require models below:

```
import torch
import pandas as pd
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
```

Python Predefine Model

Key Components

- **PyTorch** – To Fine tuning the Bio Bert model.
- **Pandas** - Used for loading and manipulating tabular data.
- **Transformers :**

  BertTokenizer: Converts raw text into token IDs suitable for BERT.

  BertForSequenceClassification: Pretrained BERT model adapted for classification tasks.

  Trainer: High-level API to train and evaluate models.

  TrainingArguments: Configuration for training (e.g., batch size, learning rate).
- **Scikit-learn: sklearn.model_selection**:To used for Data Splitting - Splits data into training and testing sets to evaluate model performance.

Implementation Phase

The Python code below explains how to split the text data into train text & test text, how to load & pretrained BioBert model, how to passing the arguments by using TrainingAruguments(), how to perform the Compute metrics(), Trainer(), Evaluation() & Prediction().

The below attached below **Python Snippet code 1** explains about:

Split Data:

This splits your dataset (data) into:

- **80% training data**
- **20% validation data**
- **TextDataset**: is a custom class that wraps the text and labels into a format suitable for PyTorch training.

 # Load Pretrained Bio BERT Model

- **num_labels:** Number of unique output classes (e.g., drug-symptom, gene-disease).
- **device:** Uses GPU if available, otherwise CPU.
- **BertForSequenceClassification:** Loads Bio BERT with a classification head for predicting relationships.

 # Set Training Arguments
Defines how the model will be trained:

- **Batch size**: 8 samples per device
- **Epochs**: Train for 3 full passes over the data
- **Save strategy**: Save model after each epoch
- **Logging**: Store logs in ./logs
- **Reporting**: Disabled external reporting (e.g., to WandB or TensorBoard)

```
# Split data
train_texts, val_texts, train_labels, val_labels = train_test_split(
    data['description'], data['label'], test_size=0.2, random_state=42)
train_dataset = TextDataset(train_texts.tolist(), train_labels.tolist())
val_dataset = TextDataset(val_texts.tolist(), val_labels.tolist())

# Load Pretrained BioBERT Model
num_labels = len(label_encoder.classes_)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = BertForSequenceClassification.from_pretrained
                        (model_name, num_labels=num_labels).to(device)

# Training Arguments
training_args = TrainingArguments(
    output_dir='./results',
    # evaluation_strategy='epoch', # Removed this argument
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    save_strategy='epoch',
    logging_dir='./logs',
    report_to="none"
)
```

Python Snippet Code 1

The attached below **Python Snippet code 2** explains about:

Define Evaluation Metrics

This function calculates **how well the model is performing**.

- **logits:** Raw output scores from the model.
- **preds:** Final predicted class (highest score).
- Metrics computed:
- **Accuracy**: % of correct predictions.
- **Precision**: How many predicted positives were correct.
- **Recall**: How many actual positives were found.
- **F1 Score**: Balance between precision and recall.

Set Up the Trainer

This wraps everything into a **Trainer object**:

- **model:** our BioBERT model.
- **training_args:** Settings like batch size, epochs, logging.
- **train_dataset and eval_dataset:** our training and validation data.
- **compute_metrics:** The function above to evaluate performance.

Train and Evaluate the Model

- **trainer.train():** Starts the training process.
- **trainer.evaluate():** Runs the model on the validation set and returns metrics.
- **print(...):** Displays the results (accuracy, precision, recall, F1).

```python
# Compute metrics
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = torch.argmax(torch.tensor(logits), dim=-1)
    acc = accuracy_score(labels, preds)
    precision, recall, f1, _ = precision_recall_fscore_support
                                            (labels, preds, average='weighted')
    return {"accuracy": acc," precision": precision, "recall": recall, "f1": f1}

# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics
)

# Train model
trainer.train()

# Evaluate model
metrics = trainer.evaluate()
print("Evaluation Metrics:", metrics)
```

Python Snippet Code 2

The attached below **Python Snippet Code 3** explains about:

Prediction Function: Tokenize the Input Text

- Converts the **input text** into tokens that BioBERT understands.
- **return_tensors="pt"** makes it compatible with PyTorch.
- **truncation, padding, and max_length** ensure consistent input size.
- Sends the tokenized inputs to GPU or CPU, depending on availability.
- **outputs.logits** contains raw scores for each class.
- Applies **softmax** to turn raw scores into probabilities.
- Picks the class with the highest probability.
- Uses **label_encoder** to map the numeric prediction back to a human-readable label (like "drug–symptom").

```
# Prediction Function
def predict_classification(text):
    inputs = tokenizer(str(text), return_tensors="pt", truncation=True, padding=True, max_length=128)
    inputs = {key: val.to(device) for key, val in inputs.items()}  # Move inputs to same device as model
    outputs = model(**inputs)
    probs = torch.nn.functional.softmax(outputs.logits, dim=-1)
    prediction = torch.argmax(probs, dim=-1).item()
    return label_encoder.inverse_transform([prediction])[0]
```

Python Snippet Code 3

Reports Level Input & Predicted Output Analysis

If we upload large volume of unstructured text data file in the format of csv, into this Bio Bert model, we are getting the below kind of output like Adverse Event / Non Event.

The attached below **predicted output using Bio Bert** Snippet shows the result of applying a **BioBERT-based text classification model** to a dataset of biomedical event descriptions. Each row represents a clinical sentence involving a drug, and the model predicts whether the sentence describes an **adverse event (AE)** or a **non-event**.

The table includes:

- **description**: A clinical or patient-reported sentence.
- **Drug**: The drug mentioned in the sentence.
- **classification**: The true label (ground truth).
- **predicted_label**: The label predicted by the model.

This output is useful for evaluating the model's performance in identifying adverse events from biomedical text. It highlights both correct predictions and misclassifications, which can inform further tuning, error analysis, or dataset refinement. It's a practical demonstration of how BioBERT can be used for **automated pharmacovigilance** or **clinical safety monitoring**.

```
1] df=pd.read_csv("event_classification_sample.csv", encoding='latin-1')
```

```
df['predicted_label'] = df['description'].apply(predict_classification)
df
```

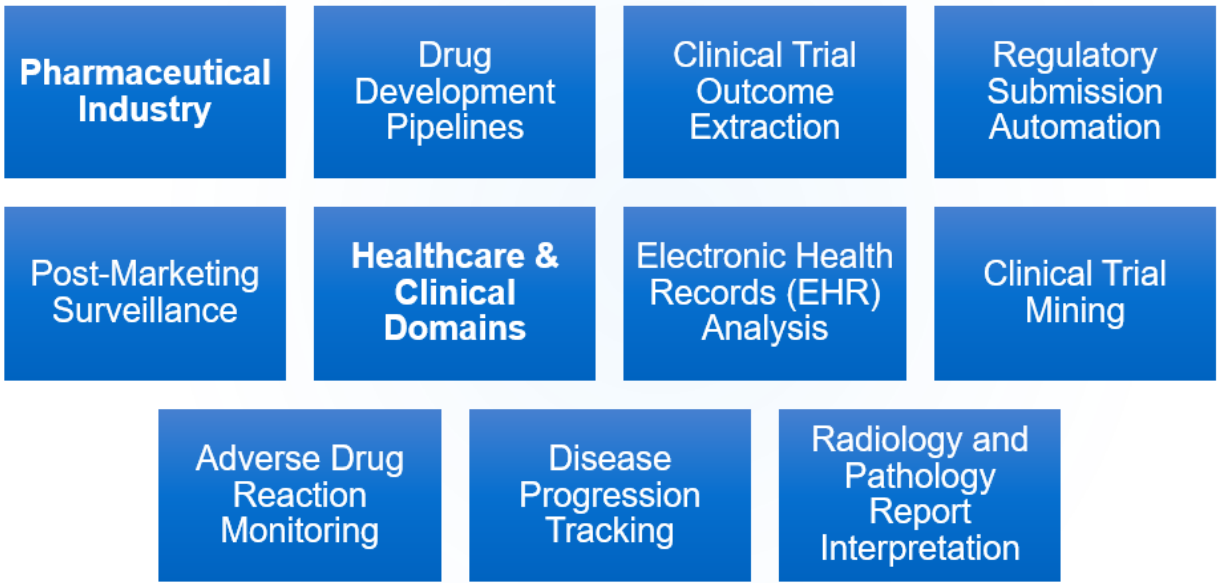
1 to 15 of 15 entries Filter

index	description	Drug	classification	predicted_label
0	while taking Drug- A the patient was diagnosed with acute myocardial infarction.	Drug A	AE	Non Event
1	No significant abnormalities were found in the blood test results.	Drug B	Non Event	AE
2	The clinical trial showed promising results for the new drug.	Drug A	Non Event	Non Event
3	The patient experienced severe chest pain and shortness of breath.	Drug B	AE	AE
4	Trying to sleep on with Drug A is so weird. There are so many jitters	Drug A	Non Event	AE
5	I take my Drug B like 2 hours before I get up for the day and like those last couple hours of sleep are always nightmares	Drug B	AE	AE
6	The patient was admitted to the hospital for observation.	Drug A	AE	Non Event

Predicted Output using Bio Bert

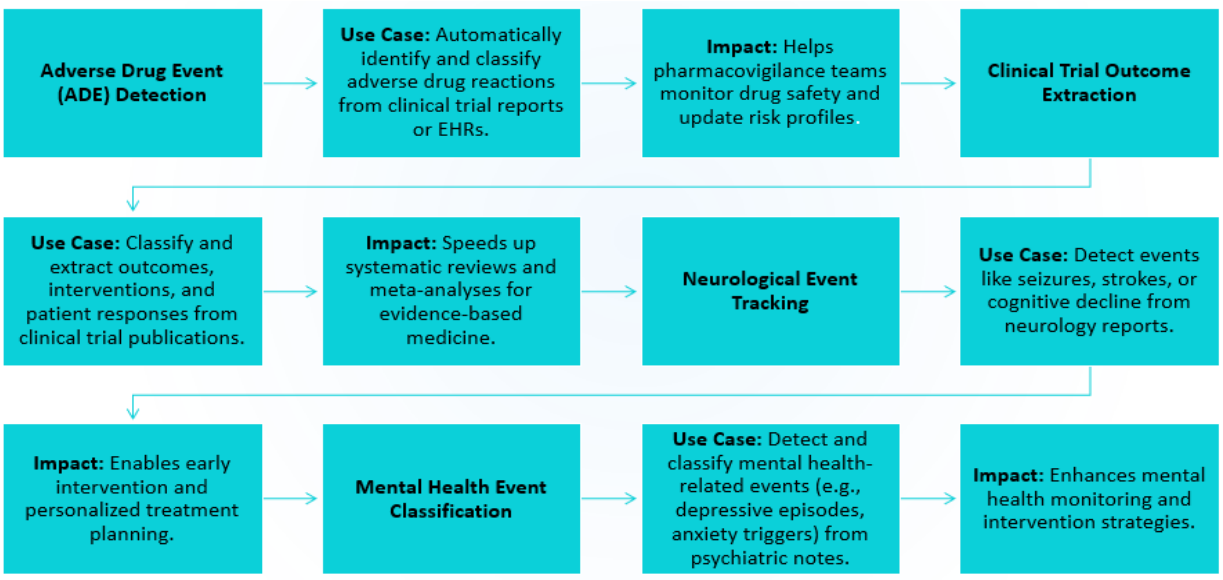
Clinical Applications of BioBERT

This Bio BERT model can be applied in multiple domains. In this paper, I have attached only few fields like Pharmaceutical Industry, Healthcare & clinical Domains.



Real World Applications of Bio Bert

Attached few clinical related applications with use case & impact as well for our more understanding.



Real World Applications of Bio Bert

Conclusion

BioBERT acts as a powerful feature extractor for biomedical text, and a subsequent classification layer uses these features to categorize events, leveraging BioBert's deep understanding of the biomedical domain.

Event classification using BioBERT involves training a biomedical language model to identify and categorize relationships or events—like adverse drug reactions—from unstructured clinical text.

This approach is particularly useful for pharmacovigilance, AE analysis, automating structured data extraction, Disease Progression and Treatment Monitoring , Clinical Trial Recruitment & Automated Coding from clinical narratives.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Gayathiri Premkumar
Company: SENSAN Biosciences
Address: Chennai, India
Email: gayathiri.premkumar@sensanbio.com
Website: www.sensanbio.com