Important!! This notebook was set to run in a Kaggle Environment

Libraries

```python
import json
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import wandb
wandb.init(project="my_project_name", mode="disabled")
```

Config

```python
!pip install evaluate
!pip install wandb
```

```python
import torch
import torch.nn as nn
from datasets import DatasetDict, Dataset, load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer, DataCollatorWithPadding
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import f1_score, roc_auc_score, precision_score, accuracy_score, recall_score
from sklearn.utils.class_weight import compute_class_weight
import evaluate
```

Datasets

Inference batch

```python
# Define file path
file_name = "test_unlabeled.tsv"
final_path = os.path.join("/kaggle/input/579nlp-project2", file_name)

# Load tsv file
inference_batch = pd.read_csv(final_path, sep='\t')
print(f"The inference batch has {inference_batch.shape[0]} observations and {inference_batch.shape[1]} columns.")
inference_batch.head()
```

```
The inference batch has 1097 observations and 4 columns.
```

	PMID	Title	Abstract	Label
0	34902587	Detection of porcine circovirus type 3 DNA in ...	Porcine circovirus type 3 (PCV3) is regularly ...	0
1	35451025	Imputation of non-genotyped F1 dams to improve...	This study investigated using imputed genotype...	0
2	34859764	Proposed multidimensional pain outcome methodo...	Castration of male piglets in the United State...	0
3	35143972	Nanostructured lipid carriers loaded with an a...	Alopecia is a condition associated with differ...	0
4	34872491	Genome-wide expression of the residual lung re...	BACKGROUND: Acute or chronic irreversible resp...	0

Training Corpus

```python
# Define file path
file_name = "QTL_text.json"
final_path = os.path.join("/kaggle/input/579nlp-project2", file_name)

# Load json file
df = pd.read_json(final_path)
df = df.drop(columns=['Journal'])
print(f"Shape of the original dataset: {df.shape}", "\n")
df.head()
```

```
Shape of the original dataset: (11278, 4)
```

	PMID	Title	Abstract	Category
0	17179536	Variance component analysis of quantitative tr...	In a previous study, QTL for carcass compositi...	1
1	17177700	Single nucleotide polymorphism identification,...	Pituitary adenylate cyclase-activating polypep...	0
2	17129674	Genetic resistance to Sarcocystis miescheriana...	Clinical and parasitological traits of Sarcocy...	0
3	17121599	Results of a whole-genome quantitative trait l...	A whole-genome quantitative trait locus (QTL) ...	1
4	17057239	Unexpected high polymorphism at the FABP4 gene...	Fatty acid bing protein 4 (FABP4) plays a key ...	0

Modeling Exploratory

Pre-Processing

```python
df_test = df.copy()
df_test['Title_Abstract'] = df_test['Title'] + " " + df_test['Abstract']
print(df.shape)
df_test.head()
```

```
(11278, 4)
```

	PMID	Title	Abstract	Category	Title_Abstract
0	17179536	Variance component analysis of quantitative tr...	In a previous study, QTL for carcass compositi...	1	Variance component analysis of quantitative tr...
1	17177700	Single nucleotide polymorphism identification,...	Pituitary adenylate cyclase-activating polypep...	0	Single nucleotide polymorphism identification,...
2	17129674	Genetic resistance to Sarcocystis miescheriana...	Clinical and parasitological traits of Sarcocy...	0	Genetic resistance to Sarcocystis miescheriana...
3	17121599	Results of a whole-genome quantitative trait l...	A whole-genome quantitative trait locus (QTL) ...	1	Results of a whole-genome quantitative trait l...
4	17057239	Unexpected high polymorphism at the FABP4 gene...	Fatty acid bing protein 4 (FABP4) plays a key ...	0	Unexpected high polymorphism at the FABP4 gene...

Stopwords

```python
# pip install nltk
```

```python
# import nltk
# from nltk.corpus import stopwords
# nltk.download("stopwords")
# stop_words = set(stopwords.words("english"))

# df_test["Title_Abstract"] = df_test["Title_Abstract"].apply(lambda x: " ".join([word for word in x.split() if word.lower() not in stop_words]))
```

Punctuation

```python
# import re

# # Define a regex string to match punctuation
# regex = r"[^\w\s]|_"

# # Define a lambda function
# remove_punct = lambda text: re.sub(regex, " ", text)

# # Apply the remove_punct function to the column
# df_test["Title_Abstract"] = df_test["Title_Abstract"].apply(remove_punct)
```

Train-Test Split

```python
# Define predictor and target features
X = df_test.drop(columns=['Category'])
y = df_test['Category']

# Split train and validation
X_train, X_val, y_train, y_val = train_test_split(X,y, test_size=.2, shuffle=True, random_state=42, stratify=y)
```

```python
# Training Data
train_data = {"text": X_train['Title_Abstract'], "labels": y_train}
train_dataset = Dataset.from_dict(train_data)

# Validation Data
val_data = {"text": X_val['Title_Abstract'], "labels": y_val}
val_dataset = Dataset.from_dict(val_data)

# # Test Data
# test_data = {"text": X_test['Title_Abstract'], "labels": y_test}
# test_dataset = Dataset.from_dict(test_data)

dataset_dict = DatasetDict({
    "train": train_dataset,
    "validation": val_dataset,
    # "test": test_dataset
})

dataset_dict
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'labels'],
        num_rows: 9022
    })
    validation: Dataset({
        features: ['text', 'labels'],
        num_rows: 2256
    })
})
```

Load Pre-Trained Model

Fine-Tune From Scratch

```python
# Define pre-trained model path
model_path = "google-bert/bert-base-uncased"

# Load model tokeninzer
tokenizer = AutoTokenizer.from_pretrained(model_path)

# Load model with binary classification head
model = AutoModelForSequenceClassification.from_pretrained(model_path,
                                                           num_labels=2,)
```

Fine-Tune From a Checkpoint

```python
# !cp -r /kaggle/input/checkpoint-14432 /kaggle/working/experiment_outputs/ # Move dataset to the working session
```

```python
# # Define pre-trained model path
# model_path = "experiment_outputs/checkpoint-14432"

# # Load model tokeninzer
# tokenizer = AutoTokenizer.from_pretrained(model_path)

# # Load model with binary classification head
# model = AutoModelForSequenceClassification.from_pretrained(model_path,
#                                                            num_labels=2,)
```

Use BERT Tokenizer to Pre-Process the Data

```python
# Define text preprocessing
def preprocess_function(examples):
    # Return tokenized text with truncation
    return tokenizer(
        examples['text'],
        truncation=True) # Truncate abstracts greater than 512 tokens

# Preprocess all datasets
tokenized_data = dataset_dict.map(preprocess_function, batched=True)

# Create data collator
data_collator = DataCollatorWithPadding(tokenizer=tokenizer) # Uniform sample lenght
```

Put Model in CUDA Mode

```python
print(torch.cuda.is_available())
```

```
True
```

```python
model = model.to('cuda')
```

Define Evaluation Metrics

```python
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    # Convert logits to probabilities using softmax for two-class classification.
    probs = torch.softmax(torch.tensor(logits), dim=1).numpy()
    # Convert to 1D binary predictions by taking the probability of the positive class and thresholding.
    threshold = 0.4  # adjust as needed
    preds = (probs[:, 1] > threshold).astype(int)

    # Ensure labels are 1D
    labels = labels.flatten() if labels.ndim > 1 else labels

    # Compute metrics
    f1 = f1_score(labels, preds, average="binary")  # This is a binary F1, not macro.
    auc = roc_auc_score(labels, probs[:, 1])
    precision = precision_score(labels, preds, average="binary")
    accuracy = accuracy_score(labels, preds)
    recall = recall_score(labels, preds, average="binary")

    return {"f1_harmonic": f1, "recall": recall, "precision": precision, "accuracy": accuracy, "auc": auc}
```

Training Parameters

```python
train_labels = np.array(train_dataset["labels"])

# Compute weights.
class_weights = compute_class_weight(class_weight="balanced", classes=np.unique(train_labels), y=train_labels)
class_weights = torch.tensor(class_weights, dtype=torch.float)
# class_weights = torch.tensor([1.5490, 3.0], dtype=torch.float)

print("Class weights:", class_weights)
```

```
Class weights: tensor([0.5491, 5.5968])
```

```python
class WeightedLossTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
        labels = inputs.get("labels")
        outputs = model(**inputs)
        logits = outputs.get("logits")
        # Use weighted CrossEntropyLoss
        loss_fct = nn.CrossEntropyLoss(weight=class_weights.to(logits.device))
        loss = loss_fct(logits.view(-1, self.model.config.num_labels), labels.view(-1))
        return (loss, outputs) if return_outputs else loss
```

Define Hyper-Parameters & Fine-Tune Model

```python
# !rm -rf /kaggle/working/experiment_outputs/checkpoint-564 # Delete a folder or file from the work session
```

```python
from transformers import EarlyStoppingCallback
```

```python
# Hyperparameters
lr = 2e-6
batch_size =16
num_epochs = 20
weight_decay=0.01

os.makedirs("experiment_outputs", exist_ok=True)

training_args = TrainingArguments(
    output_dir="experiment_outputs",
    learning_rate=lr,
    weight_decay=weight_decay,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=num_epochs,
    logging_strategy="epoch",
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    evaluation_strategy="epoch",
    metric_for_best_model="eval_loss",  # Metric to monitor
    greater_is_better=False,
)

trainer_weighted = WeightedLossTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_data['train'],
    eval_dataset=tokenized_data['validation'],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

trainer_weighted.train() # Train from scratch
# trainer_weighted.train(resume_from_checkpoint=model_path) # Continue training from a specific checkpoint
# trainer_weighted.train(resume_from_checkpoint=True) # Continue training from the latest checkpoint
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1575: FutureWarning: `evaluation_strategy` is deprecated and will be removed in versio
n 4.46 of 🤗 Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-78-cb463a958f53>:25: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `WeightedLossTrainer.__init__`. Use `p
rocessing_class` instead.
  trainer_weighted = WeightedLossTrainer(
/usr/local/lib/python3.10/dist-packages/transformers/trainer.py:3418: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defau
lt value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpic
kling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_
only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loade
d via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=
True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feat
ure.
  torch.load(os.path.join(checkpoint, OPTIMIZER_NAME), map_location=map_location)
/usr/local/lib/python3.10/dist-packages/transformers/trainer.py:3081: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defau
lt value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpic
kling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_
only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loade
d via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=
True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feat
ure.
  checkpoint_rng_state = torch.load(rng_file)
```

[11280/11280 44:29, Epoch 20/20]

Epoch	Training Loss	Validation Loss	F1 Harmonic	Recall	Precision	Accuracy	Auc
16	0.138600	0.214811	0.829694	0.945274	0.739300	0.965426	0.987421
17	0.137400	0.211397	0.814499	0.950249	0.712687	0.961436	0.988062
18	0.142400	0.216294	0.817987	0.950249	0.718045	0.962323	0.987997
19	0.124200	0.221191	0.824295	0.945274	0.730769	0.964096	0.987839
20	0.117000	0.220263	0.822511	0.945274	0.727969	0.963652	0.987759

```
Could not locate the best model at experiment_outputs/checkpoint-1128/pytorch_model.bin, if you are running a distributed training on multiple nodes, you sh
ould activate `--save_on_each_node`.
```

```
TrainOutput(global_step=11280, training_loss=0.03297780151908279, metrics={'train_runtime': 2670.6188, 'train_samples_per_second': 67.565, 'train_steps_per
_second': 4.224, 'total_flos': 4.733871908041824e+16, 'train_loss': 0.03297780151908279, 'epoch': 20.0})
```

Test Data

```python
# # Apply model to validation dataset
# predictions = trainer_weighted.predict(tokenized_data["test"])

# # Extract the logits and labels from the predictions object
# logits = predictions.predictions
# labels = predictions.label_ids

# # Compute metrics
# metrics = compute_metrics((logits, labels))
# print(metrics)
```

```
{'f1': 1.0, 'auc': 1.0, 'precision': 1.0, 'accuracy': 1.0}
```

Save Checkpoint

```python
import shutil

checkpoint = 'checkpoint-9588'

# Compress the checkpoint folder into a zip file.
shutil.make_archive(f'/kaggle/working/experiment_outputs/{checkpoint}', 'zip', f'/kaggle/working/experiment_outputs/{checkpoint}')
```

```
'/kaggle/working/experiment_outputs/checkpoint-9588.zip'
```

```python
from IPython.display import FileLink, display

FileLink(f"experiment_outputs/{checkpoint}.zip")
```

experiment_outputs/checkpoint-9588.zip

Use Model to Run Predictions on the Inference Data

```python
import torch
torch.cuda.empty_cache()
```

```python
inference_batch['Title_Abstract'] = inference_batch['Title'] + " " + inference_batch['Abstract']
print(inference_batch.shape)
inference_batch.head()
```

```
(1097, 5)
```

	PMID	Title	Abstract	Label	Title_Abstract
0	34902587	Detection of porcine circovirus type 3 DNA in ...	Porcine circovirus type 3 (PCV3) is regularly ...	1	Detection of porcine circovirus type 3 DNA in ...
1	35451025	Imputation of non-genotyped F1 dams to improve...	This study investigated using imputed genotype...	1	Imputation of non-genotyped F1 dams to improve...
2	34859764	Proposed multidimensional pain outcome methodo...	Castration of male piglets in the United State...	0	Proposed multidimensional pain outcome methodo...
3	35143972	Nanostructured lipid carriers loaded with an a...	Alopecia is a condition associated with differ...	0	Nanostructured lipid carriers loaded with an a...
4	34872491	Genome-wide expression of the residual lung re...	BACKGROUND: Acute or chronic irreversible resp...	1	Genome-wide expression of the residual lung re...

```python
device = torch.device("cpu")
model = model.to(device)
```

```python
# Tokenize texts
tokenized_inputs = tokenizer(
    inference_batch['Title_Abstract'].tolist(),
    truncation=True,
    padding=True,
    return_tensors="pt"
)


# Move tokenized inputs to the model's device
device = model.device
tokenized_inputs = {key: value.to(device) for key, value in tokenized_inputs.items()}

#  Run inference
model.eval()  # Set to evaluation mode to disable dropout, etc.
with torch.no_grad():
    outputs = model(**tokenized_inputs)

# Convert logits to probabilities using softmax
probs = torch.softmax(outputs.logits, dim=1)  # Shape: (batch_size, 2)

# Extract the probability of the positive class
positive_probs = probs[:, 1]

# Set a custom threshold
threshold = 0.4
predictions = (positive_probs > threshold).int()

# Convert predictions to numpy array
predictions = predictions.cpu().numpy()
```

```python
# Set a custom threshold
threshold = 0.0018 # Best = 0.0018
predictions = (positive_probs > threshold).int()

# Convert predictions to numpy array
predictions = predictions.cpu().numpy()
```

```python
# Attach predictions to the DataFrame
inference_batch['Label'] = predictions
print(inference_batch.shape)
inference_batch[['Title_Abstract', 'Label']].head()
```

```
(1097, 5)
```

	Title_Abstract	Label
0	Detection of porcine circovirus type 3 DNA in ...	0
1	Imputation of non-genotyped F1 dams to improve...	0
2	Proposed multidimensional pain outcome methodo...	0
3	Nanostructured lipid carriers loaded with an a...	0
4	Genome-wide expression of the residual lung re...	0

```python
inference_batch['Label'].value_counts()
```

```
Label
0    890
1    207
Name: count, dtype: int64
```

```python
inference_batch[['Title_Abstract', 'Label']].head()
```

	Title_Abstract	Label
0	Detection of porcine circovirus type 3 DNA in ...	0
1	Imputation of non-genotyped F1 dams to improve...	0
2	Proposed multidimensional pain outcome methodo...	0
3	Nanostructured lipid carriers loaded with an a...	0
4	Genome-wide expression of the residual lung re...	0

```python
inference_batch[['PMID', 'Label']].to_csv("solution_26.csv", index=False)
```