!pip install -U accelerate==0.28.0

!pip install -U transformers==4.38.2

!pip install -U evaluate==0.4.0

!pip install -U rouge\_score==0.1.2

!pip install -U peft==0.3.0 --quiet

!pip install -U transformers accelerate

import transformers

import accelerate

print(transformers.\_\_version\_\_)

print(accelerate.\_\_version\_\_)

!pip install numpy==1.26

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.utils.class\_weight import compute\_class\_weight

from sklearn.metrics import classification\_report, confusion\_matrix

!pip install peft==0.10.0

import torch

from torch.utils.data import Dataset

from transformers import (

    DebertaV2TokenizerFast,

    DebertaV2ForSequenceClassification,

    Trainer,

    TrainingArguments,

    EarlyStoppingCallback

)

import accelerate

print(accelerate.\_\_file\_\_)

df = pd.read\_csv('combined\_miss.csv')

#dftrain=pd.read\_csv('healthver\_train.csv')

#dftest=pd.read\_csv('healthver\_test.csv')

#dfdev=pd.read\_csv('healthver\_dev.csv')

print(df.head())

print(df['label'].value\_counts())

raw\_claims = df["claim"]

from datasets import Dataset, ClassLabel

# Example: your dataset

dataset = Dataset.from\_pandas(df)  # or however you made it

# Create the ClassLabel feature

features = dataset.features.copy()

features["label"] = ClassLabel(num\_classes=3, names=["false", "true", "unverified"])

# Cast the column

dataset = dataset.cast(features)

print(dataset.features)

print(dataset[:10])

print(dataset.unique('label'))

print(dataset[:10])

from sklearn.model\_selection import train\_test\_split

# Get your data as Pandas

df = dataset.to\_pandas()

# Do the split with stratify

train\_df, test\_df = train\_test\_split(

    df,

    test\_size=0.2,

    random\_state=42,

    stratify=df['label']

)

# Convert back to Dataset

from datasets import Dataset

train\_ds = Dataset.from\_pandas(train\_df)

test\_ds = Dataset.from\_pandas(test\_df)

print(train\_ds, test\_ds)

labels = df['label'].values

class\_weights = compute\_class\_weight(class\_weight='balanced', classes=np.unique(labels), y=labels)

class\_weights = torch.tensor(class\_weights, dtype=torch.float)

print("Class weights:", class\_weights)

model = DebertaV2ForSequenceClassification.from\_pretrained('microsoft/deberta-v3-base', num\_labels=3)

# Override loss function for class weights

from torch.nn import CrossEntropyLoss

def compute\_loss(model, inputs, return\_outputs=False):

    labels = inputs.get("labels")

    outputs = model(\*\*inputs)

    logits = outputs.get("logits")

    loss\_fct = CrossEntropyLoss(weight=class\_weights.to(logits.device))

    loss = loss\_fct(logits, labels)

    return (loss, outputs) if return\_outputs else loss

model.compute\_loss = compute\_loss

from accelerate import Accelerator, DataLoaderConfiguration

# Define DataLoaderConfiguration

dataloader\_config = DataLoaderConfiguration(

    dispatch\_batches=False,  # Each process fetches its own batch

    split\_batches=True       # Split fetched batches across processes

)

# Initialize Accelerator with DataLoaderConfiguration

accelerator = Accelerator(dataloader\_config=dataloader\_config)

from transformers import AutoTokenizer

# Example: DeBERTa

tokenizer = AutoTokenizer.from\_pretrained("microsoft/deberta-v3-small")

def tokenize\_fn(batch):

    return tokenizer(

        batch['claim'],

        truncation=True,

        padding='max\_length',

        max\_length=128

    )

# Apply it!

tokenized\_ds = dataset.map(tokenize\_fn, batched=True)

# Remove raw text if needed

tokenized\_ds = tokenized\_ds.remove\_columns(['claim'])

# Set format for PyTorch

tokenized\_ds.set\_format(type='torch', columns=['input\_ids', 'attention\_mask', 'label'])

# Now split

split = tokenized\_ds.train\_test\_split(test\_size=0.2, seed=42, stratify\_by\_column='label')

train\_ds = split['train']

test\_ds = split['test']

# TrainingArguments

training\_args = TrainingArguments(

    output\_dir="./results",

    eval\_strategy="epoch",

    save\_strategy="epoch",

    load\_best\_model\_at\_end=True,

    metric\_for\_best\_model="eval\_loss",

    greater\_is\_better=False,

    per\_device\_train\_batch\_size=16,

    per\_device\_eval\_batch\_size=16,

    num\_train\_epochs=5,

    report\_to="none"

)

trainer = Trainer(

    model=model,

    args=training\_args,

    train\_dataset=train\_ds,

    eval\_dataset=test\_ds

)

import os

os.environ["WANDB\_DISABLED"] = "true"

print(train\_ds.column\_names)

print(test\_ds.column\_names)

print(train\_ds[0])

trainer.train()

from sklearn.metrics import accuracy\_score, f1\_score, precision\_recall\_fscore\_support, confusion\_matrix, classification\_report

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# 1️⃣ Define compute\_metrics for Trainer

def compute\_metrics(eval\_pred):

    logits, labels = eval\_pred

    preds = np.argmax(logits, axis=1)

    precision, recall, f1, \_ = precision\_recall\_fscore\_support(labels, preds, average='macro')

    acc = accuracy\_score(labels, preds)

    return {

        'accuracy': acc,

        'macro\_f1': f1,

        'macro\_precision': precision,

        'macro\_recall': recall

    }

# 2️⃣ After training/evaluation:

# Get predictions from your trainer

preds = trainer.predict(test\_ds)

y\_true = preds.label\_ids

y\_pred = np.argmax(preds.predictions, axis=1)

# 3️⃣ Print classification report

print("\nClassification Report:")

print(classification\_report(y\_true, y\_pred, target\_names=['false', 'true', 'unverified']))

# 4️⃣ Plot confusion matrix

cm = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(6,5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

            xticklabels=['false', 'true', 'unverified'],

            yticklabels=['false', 'true', 'unverified'])

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.title('Confusion Matrix')

plt.show()

df

import numpy as np

import pandas as pd

# Assuming preds.predictions is logits for the test set

probs = torch.nn.Softmax(dim=1)(torch.tensor(preds.predictions)).numpy()

# misclassified\_mask: boolean array where predictions != true labels

misclassified\_mask = (y\_pred != y\_true)

# Get indices of misclassified samples

misclassified\_indices = np.where(misclassified\_mask)[0]

# For these misclassified samples, get predicted class probabilities

misclassified\_probs = probs[misclassified\_indices, y\_pred[misclassified\_mask]]

# Sort misclassified by ascending predicted prob (lowest confidence)

sorted\_indices = np.argsort(misclassified\_probs)

# Take top 100 most uncertain misclassified

top\_100\_indices = misclassified\_indices[sorted\_indices[:100]]

# Get corresponding rows from original dataset (reset index if needed)

top\_100\_df = df.reset\_index(drop=True).iloc[top\_100\_indices].copy()

# Add predicted labels and confidence scores

top\_100\_df['predicted\_label'] = y\_pred[top\_100\_indices]

top\_100\_df['predicted\_prob'] = misclassified\_probs[sorted\_indices[:100]]

# Display or save

print(top\_100\_df[['claim', 'label', 'predicted\_label', 'predicted\_prob']])

from IPython.display import display

# Show a clean table with important columns

display(top\_100\_df[['claim', 'label', 'predicted\_label', 'predicted\_prob']])

import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.bar(range(len(top\_100\_df)), top\_100\_df['predicted\_prob'], color='tomato')

plt.xlabel('Misclassified Sample Index')

plt.ylabel('Predicted Confidence (Probability)')

plt.title('Predicted Confidence of Top 100 Misclassified Claims')

plt.xticks(rotation=90)

plt.show()

import seaborn as sns

plt.figure(figsize=(10,5))

sns.countplot(x='predicted\_label', data=top\_100\_df)

plt.title('Distribution of Predicted Labels Among Top 100 Misclassified')

plt.show()

plt.figure(figsize=(10,5))

sns.countplot(x='label', data=top\_100\_df)

plt.title('Distribution of True Labels Among Top 100 Misclassified')

plt.show()

top\_100\_df['claim\_length'] = top\_100\_df['claim'].apply(lambda x: len(x.split()))

print("Average claim length:", top\_100\_df['claim\_length'].mean())

import pandas as pd

import numpy as np

# Original DataFrame

df = pd.read\_csv("combined\_healthver.csv")  # contains 'claim' and 'label'

# After splitting your dataset (assuming you did a simple train\_test\_split without shuffling indices)

from sklearn.model\_selection import train\_test\_split

train\_df, test\_df = train\_test\_split(df, test\_size=0.2, random\_state=42, stratify=df['label'])

# After prediction on test set using your model

# y\_true: true labels for test set, numpy array or list

# y\_pred: predicted labels for test set, numpy array or list

# Convert y\_true and y\_pred to numpy arrays (if not already)

y\_true = np.array(y\_true)

y\_pred = np.array(y\_pred)

# Get boolean array where prediction is wrong

misclassified\_mask = y\_true != y\_pred

# Extract misclassified rows from test\_df (make sure index alignment!)

misclassified\_df = test\_df.loc[misclassified\_mask]

# Add columns for predicted labels

misclassified\_df = misclassified\_df.copy()

misclassified\_df['predicted\_label'] = y\_pred[misclassified\_mask]

# Optionally save to CSV

misclassified\_df.to\_csv("misclassified\_claims.csv", index=False)

print(f"Saved {len(misclassified\_df)} misclassified claims.")

import pandas as pd

import numpy as np

df = df.reset\_index(drop=True)

import pandas as pd

# Make sure you only keep \*actually wrong\* examples

wrong = top\_100\_df[top\_100\_df["label"] != top\_100\_df["predicted\_label"]]

# Sort by lowest probability if you want the most uncertain

wrong\_sorted = wrong.sort\_values("predicted\_prob").head(100)

# Save to CSV

wrong\_sorted.to\_csv("misclassified\_top100.csv", index=False)

# Show sample

wrong\_sorted.head(5)