CS 7650 Final Project - Lexical Normalization

Import statements

```
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = "1"
os.environ['TORCH_USE_CUDA_DSA'] = "1"
from transformers import T5Tokenizer, T5ForConditionalGeneration, Trainer, TrainingArguments, BertTokenizer, DataCollatorForSeq2
from torch.utils.data import Dataset
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
import matplotlib.pyplot as plt
import torch
from tqdm import tqdm
import numpy as np
import editdistance
from collections import Counter
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
from google.colab import drive

drive.mount('/content/drive')
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
cuda
```

Process dataset

```
class LexicalNormalizationDataset(Dataset):
    def __init__(self, x, y, tokenizer):
        self.encodings = tokenize_data(x, y, tokenizer)

    def __len__(self):
        return len(self.encodings["input_ids"])

    def __getitem__(self, idx):
        return {key: val[idx] for key, val in self.encodings.items()}

class TweetNormalizationDataset(Dataset):
    def __init__(self, sources, targets, tokenizer, max_input_len=64, max_target_len=64):
        self.inputs = tokenizer(sources, truncation=True, padding=True, max_length=max_input_len, return_tensors="pt")
        with tokenizer.as_target_tokenizer():
            self.targets = tokenizer(targets, truncation=True, padding=True, max_length=max_target_len, return_tensors="pt")
        self.inputs["labels"] = self.targets["input_ids"]

    def __len__(self):
        return len(self.inputs["input_ids"])

    def __getitem__(self, idx):
        return {key: val[idx] for key, val in self.inputs.items()}


def process_data(path):
    x, y = [], []
    with open(path, 'r') as f:
        for line in f:
            parts = line.strip().split("\t", 1)
            if len(parts) == 2:
                noisy = parts[0]
                clean = parts[1]
                x.append(noisy)
                y.append(clean)
    return x, y

def process_tweet_level_data(path):
    inputs, targets = [], []
    current_input, current_target = [], []
    with open(path, 'r') as f:
        for line in f:
            if line.strip() == "":
```

```
            if current_input:
                inputs.append(" ".join(current_input))
                targets.append(" ".join(current_target))
                current_input, current_target = [], []
        else:
            parts = line.strip().split("\t", 1)
            if len(parts) == 2:
                noisy = parts[0]
                clean = parts[1]
                current_input.append(noisy)
                current_target.extend(clean.split())

    if current_input:
        inputs.append(" ".join(current_input))
        targets.append(" ".join(current_target))

    return inputs, targets

train_x, train_y = process_data("drive/MyDrive/Colab Notebooks/CS 7650/Final Project/data/train.norm")
test_x, test_y = process_data("drive/MyDrive/Colab Notebooks/CS 7650/Final Project/data/test.norm")
dev_x, dev_y = process_data("drive/MyDrive/Colab Notebooks/CS 7650/Final Project/data/dev.norm")

train_tweet_x, train_tweet_y = process_tweet_level_data("drive/MyDrive/Colab Notebooks/CS 7650/Final Project/data/train.norm")
dev_tweet_x, dev_tweet_y = process_tweet_level_data("drive/MyDrive/Colab Notebooks/CS 7650/Final Project/data/dev.norm")
test_tweet_x, test_tweet_y = process_tweet_level_data("drive/MyDrive/Colab Notebooks/CS 7650/Final Project/data/test.norm")
```

Part 1: T5 Model

Tokenizer

```
tokenizer = T5Tokenizer.from_pretrained("t5-small")

def tokenize_data(x, y, tokenizer, max_input_len=16, max_target_len=16):
    model_inputs = tokenizer(x, padding=True, truncation=True, max_length=max_input_len, return_tensors="pt")
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(y, padding=True, truncation=True, max_length=max_target_len, return_tensors="pt")
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs

train_dataset = LexicalNormalizationDataset(train_x, train_y, tokenizer)
test_dataset = LexicalNormalizationDataset(test_x, test_y, tokenizer)
dev_dataset = LexicalNormalizationDataset(dev_x, dev_y, tokenizer)

train_tweet_dataset = TweetNormalizationDataset(train_tweet_x, train_tweet_y, tokenizer)
test_tweet_dataset = TweetNormalizationDataset(test_tweet_x, test_tweet_y, tokenizer)
dev_tweet_dataset = TweetNormalizationDataset(dev_tweet_x, dev_tweet_y, tokenizer)
```

```
tokenizer_config.json: 100%                                  2.32k/2.32k [00:00<00:00, 221kB/s]

spiece.model: 100%                                        792k/792k [00:00<00:00, 3.63MB/s]

tokenizer.json: 100%                                      1.39M/1.39M [00:00<00:00, 6.28MB/s]

You are using the default legacy behaviour of the <class 'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is expe
/usr/local/lib/python3.11/dist-packages/transformers/tokenization_utils_base.py:3980: UserWarning: `as_target_tokenizer` is
  warnings.warn(
```

Train model

```
model = T5ForConditionalGeneration.from_pretrained("t5-small").to(device)

model_tweet = T5ForConditionalGeneration.from_pretrained("t5-small").to(device)

training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    learning_rate=3e-4,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    num_train_epochs=3,
    weight_decay=0.01,
    save_strategy="epoch",
    fp16=True,
    logging_dir="./logs",
    report_to="none",
)

training_tweet_args = TrainingArguments(
    output_dir="./tweet-results",
    eval_strategy="epoch",
    learning_rate=3e-4,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=8,
    weight_decay=0.01,
    save_strategy="epoch",
    fp16=True,
    logging_dir="./tweet-logs",
    report_to="none",
)


trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=dev_dataset,
    tokenizer=tokenizer
)

trainer_tweet = Trainer(
    model=model,
    args=training_tweet_args,
    train_dataset=train_tweet_dataset,
    eval_dataset=dev_tweet_dataset,
    tokenizer=tokenizer
)

trainer.train()

trainer_tweet.train()
```

```
config.json: 100%                                        1.21k/1.21k [00:00<00:00, 151kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falli
model.safetensors: 100%                                        242M/242M [00:00<00:00, 280MB/s]
generation_config.json: 100%                                        147/147 [00:00<00:00, 16.5kB/s]
<ipython-input-30-dbdf6ff16e56>:34: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Train
  trainer = Trainer(
<ipython-input-30-dbdf6ff16e56>:42: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Train
  trainer_tweet = Trainer(
Passing a tuple of `past_key_values` is deprecated and will be removed in Transformers v4.48.0. You should pass an instance
[3303/3303 07:45, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss
1	0.026500	0.016761
2	0.018400	0.014307
3	0.014700	0.013604

```
[2360/2360 05:36, Epoch 8/8]
```

Epoch	Training Loss	Validation Loss
1	No log	0.061786
2	0.072800	0.058348
3	0.072800	0.057732
4	0.045100	0.058353
5	0.045100	0.056768
6	0.032900	0.057863
7	0.024700	0.060176
8	0.024700	0.060578

```
TrainOutput(global_step=2360, training_loss=0.040457615609896386, metrics={'train_runtime': 336.2808,
'train_samples_per_second': 56.144, 'train_steps_per_second': 7.018, 'total_flos': 319406651473920.0, 'train_loss':
0.040457615609896386, 'epoch': 8.0})
```

Evaluation and Inference

```
def generate_predictions(model, tokenizer, inputs, batch_size=32, max_length=64):
    model.eval()
    preds = []
    for i in tqdm(range(0, len(inputs), batch_size)):
        batch_inputs = inputs[i:i+batch_size]
        encodings = tokenizer(batch_inputs, return_tensors="pt", padding=True, truncation=True).to(device)
        with torch.no_grad():
            outputs = model.generate(**encodings, max_length=max_length)
        decoded = tokenizer.batch_decode(outputs, skip_special_tokens=True)
        preds.extend([pred.strip() for pred in decoded])
    return preds

test_preds = generate_predictions(model, tokenizer, test_x)
test_tweet_preds = generate_predictions(model_tweet, tokenizer, test_tweet_x)
```

```
100%|██████████| 919/919 [03:55<00:00,  3.90it/s]
100%|██████████| 62/62 [01:08<00:00,  1.11s/it]
```

```
# Save predictions

output_dir = 'drive/MyDrive/Colab Notebooks/CS 7650/Final Project'
test_preds_file = os.path.join(output_dir, 'test_preds.txt')
test_tweet_preds_file = os.path.join(output_dir, 'test_tweet_preds.txt')

with open(test_preds_file, 'w') as f:
    for pred in test_preds:
        f.write(pred + '\n')

with open(test_tweet_preds_file, 'w') as f:
    for pred in test_tweet_preds:
        f.write(pred + '\n')

print(f"Predictions saved to: {test_preds_file} and {test_tweet_preds_file}")
```

```
Predictions saved to: drive/MyDrive/Colab Notebooks/CS 7650/Final Project/test_preds.txt and drive/MyDrive/Colab Notebooks/C
```

```
def compute_bleu(preds, refs):
    smoothie = SmoothingFunction().method4
    scores = [
        sentence_bleu([ref.split()], pred.split(), weights=(1, 0, 0, 0), smoothing_function=smoothie)
        for pred, ref in zip(preds, refs)
    ]
    return sum(scores) / len(scores)

def average_edit_distance(preds, refs):
    dists = [editdistance.eval(p, r) for p, r in zip(preds, refs)]
    return np.mean(dists)

def get_top_error_sources(inputs, preds, refs, top_n=10):
    errors = [inp for inp, pred, ref in zip(inputs, preds, refs) if pred.strip() != ref.strip()]
    counter = Counter(errors)
    return counter.most_common(top_n)

def compute_metrics(preds, labels):
    preds = [p.strip() for p in preds]
    labels = [l.strip() for l in labels]

    acc = accuracy_score(labels, preds)
    bleu = compute_bleu(preds, labels)
    edit_dist = average_edit_distance(preds, labels)
    top_error_sources = get_top_error_sources(test_x, preds, labels)

    return {
        "accuracy": acc,
        "bleu": bleu,
        "edit_distance": edit_dist,
        "top_error_sources": top_error_sources
    }

metrics = compute_metrics(test_preds, test_y)
print("T5 Model Metrics:")
print("Accuracy: ", metrics["accuracy"])
print("BLEU: ", metrics["bleu"])
print("Average Edit Distance: ", metrics["edit_distance"])
print("Top 10 Error Sources:")
for source, count in metrics["top_error_sources"]:
    print(f"Word: {source}, Count: {count}")

metrics_tweet = compute_metrics(test_tweet_preds, test_tweet_y)
print("T5 Tweet Model Metrics:")
print("Accuracy: ", metrics_tweet["accuracy"])
print("BLEU: ", metrics_tweet["bleu"])
print("Average Edit Distance: ", metrics_tweet["edit_distance"])
print("Top 10 Error Sources:")
for source, count in metrics_tweet["top_error_sources"]:
    print(f"Word: {source}, Count: {count}")
```

```
T5 Model Metrics:
Accuracy:  0.961501836484832
BLEU:  0.9618639887909209
Average Edit Distance:  0.12872398313154673
Top 10 Error Sources:
Word: 2, Count: 26
```

```
Word: rt, Count: 20
Word: ya, Count: 15
Word: b, Count: 11
Word: d, Count: 9
Word: tha, Count: 8
Word: ur, Count: 7
Word: yo, Count: 7
Word: tf, Count: 7
Word: y, Count: 7
T5 Tweet Model Metrics:
Accuracy:  0.01677681748856126
BLEU:  0.504773904913598
Average Edit Distance:  41.24148449415353
Top 10 Error Sources:
Word: :, Count: 53
Word: rt, Count: 41
Word: ., Count: 33
Word: ,, Count: 30
Word: i, Count: 29
Word: and, Count: 27
Word: the, Count: 27
Word: to, Count: 26
Word: a, Count: 22
Word: you, Count: 18
```

Part 2: BERT + GPT Model

Model Architecture

```python
def tokenize_tweet_pairs(sources, targets, max_input_len=64, max_target_len=64):
    encoder_inputs = bert_tokenizer(sources, padding="max_length", truncation=True,
                                max_length=max_input_len, return_tensors="pt")

    decoder_inputs = gpt2_tokenizer(targets, padding="max_length", truncation=True,
                                max_length=max_target_len, return_tensors="pt")

    labels = decoder_inputs["input_ids"]
    labels[labels == gpt2_tokenizer.pad_token_id] = -100

    return {
        "input_ids": encoder_inputs["input_ids"],
        "attention_mask": encoder_inputs["attention_mask"],
        "decoder_input_ids": decoder_inputs["input_ids"],
        "labels": labels
    }

class TweetSeq2SeqDataset(Dataset):
    def __init__(self, sources, targets):
        self.data = tokenize_tweet_pairs(sources, targets)

    def __len__(self):
        return len(self.data["input_ids"])

    def __getitem__(self, idx):
        return {key: val[idx] for key, val in self.data.items()}

bert_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

gpt2_tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
# tell GPT2 that it now has a pad token (we'll reuse its eos token)
gpt2_tokenizer.pad_token = gpt2_tokenizer.eos_token

# 2) Encoder–Decoder via helper API
bert_gpt_model = EncoderDecoderModel.from_encoder_decoder_pretrained(
    "bert-base-uncased",    # encoder
    "gpt2",                 # decoder
    # (from_encoder_decoder_pretrained will automatically set is_decoder=True & add_cross_attention
)

# 3) Resize decoder embeddings after adding pad_token
bert_gpt_model.decoder.resize_token_embeddings(len(gpt2_tokenizer))

# 4) Config tweaks
bert_gpt_model.config.decoder_start_token_id = gpt2_tokenizer.bos_token_id
bert_gpt_model.config.eos_token_id           = gpt2_tokenizer.eos_token_id
bert_gpt_model.config.pad_token_id           = gpt2_tokenizer.pad_token_id
# ensure the vocab_size is right for the LM head
# 5) Datasets (unchanged)
train_bert_dataset = TweetSeq2SeqDataset(train_x, train_y)
dev_bert_dataset   = TweetSeq2SeqDataset(dev_x,   dev_y)

# 6) Data collator for proper seq2seq padding & label masking
data_collator = DataCollatorForSeq2Seq(
    tokenizer=gpt2_tokenizer,
    model=bert_gpt_model,
    label_pad_token_id=-100,
```

```
Some weights of GPT2LMHeadModel were not initialized from the model checkpoint at gpt2 and are newly initialized: ['transfor
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
training_args = TrainingArguments(
    output_dir="./bert2gpt-results",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=3e-4,
    per_device_train_batch_size=64,
    per_device_eval_batch_size=64,
    num_train_epochs=3,
    weight_decay=0.01,
    fp16=True,
    logging_dir="./bert2gpt-logs",
    report_to="none"
)

trainer = Trainer(
    model=bert_gpt_model,
    args=training_args,
    train_dataset=train_bert_dataset,
    eval_dataset=dev_bert_dataset,
    data_collator=data_collator,
)

trainer.train()
```

```
/usr/local/lib/python3.11/dist-packages/transformers/models/encoder_decoder/modeling_encoder_decoder.py:651: FutureWarning:
  warnings.warn(DEPRECATION_WARNING, FutureWarning)
```

[1653/1653 17:15, Epoch 3/3]

Epoch	Training Loss	Validation Loss
1	1.963800	1.724527
2	0.968500	1.332911
3	0.415300	1.343295

```
/usr/local/lib/python3.11/dist-packages/transformers/models/encoder_decoder/modeling_encoder_decoder.py:651: FutureWarning:
  warnings.warn(DEPRECATION_WARNING, FutureWarning)
/usr/local/lib/python3.11/dist-packages/transformers/models/encoder_decoder/modeling_encoder_decoder.py:651: FutureWarning:
  warnings.warn(DEPRECATION_WARNING, FutureWarning)
TrainOutput(global_step=1653, training_loss=1.035993953365596, metrics={'train_runtime': 1036.3339,
'train_samples_per_second': 101.906, 'train_steps_per_second': 1.595, 'total_flos': 8073033403760640.0, 'train_loss':
1.035993953365596, 'epoch': 3.0})
```

```
def generate_predictions(model, encoder_tokenizer, decoder_tokenizer, inputs, batch_size: int = 64, max_length: int = 64):
    model.eval()
    all_preds = []
    for i in tqdm(range(0, len(inputs), batch_size)):
        batch = inputs[i : i + batch_size]
        enc = encoder_tokenizer(batch, return_tensors="pt", padding="longest", truncation=True, max_length=32).to(model.device)
        with torch.no_grad():
            out_ids = model.generate(input_ids=enc.input_ids, attention_mask=enc.attention_mask, max_length=max_length, early_st
        decoded = decoder_tokenizer.batch_decode(out_ids, skip_special_tokens=True)
        all_preds.extend([d.strip() for d in decoded])
    return all_preds

test_bert_preds = generate_predictions(bert_gpt_model, bert_tokenizer, gpt2_tokenizer, test_x[:250], max_length=2)
metrics_bert = compute_metrics(test_bert_preds, test_y[:250])
```