DL4H_Team_137

April 14, 2024

1 Before you use this template

This template is just a recommended template for project Report. It only considers the general type of research in our paper pool. Feel free to edit it to better fit your project. You will iteratively update the same notebook submission for your draft and the final submission. Please check the project rubriks to get a sense of what is expected in the template.

---

2 FAQ and Attentions

- Copy and move this template to your Google Drive. Name your notebook by your team ID (upper-left corner). Don't eidt this original file.
- This template covers most questions we want to ask about your reproduction experiment. You don't need to exactly follow the template, however, you should address the questions. Please feel free to customize your report accordingly.
- any report must have run-able codes and necessary annotations (in text and code comments).
- The notebook is like a demo and only uses small-size data (a subset of original data or processed data), the entire runtime of the notebook including data reading, data process, model training, printing, figure plotting, etc, must be within 8 min, otherwise, you may get penalty on the grade.
  - If the raw dataset is too large to be loaded you can select a subset of data and pre-process the data, then, upload the subset or processed data to Google Drive and load them in this notebook.
  - If the whole training is too long to run, you can only set the number of training epoch to a small number, e.g., 3, just show that the training is runable.
  - For results model validation, you can train the model outside this notebook in advance, then, load pretrained model and use it for validation (display the figures, print the metrics).
- The post-process is important! For post-process of the results,please use plots/figures. The code to summarize results and plot figures may be tedious, however, it won't be waste of time since these figures can be used for presentation. While plotting in code, the figures should have titles or captions if necessary (e.g., title your figure with "Figure 1. xxxx")
- There is not page limit to your notebook report, you can also use separate notebooks for the report, just make sure your grader can access and run/test them.
- If you use outside resources, please refer them (in any formats). Include the links to the resources if necessary.

3 Mount Notebook to Google Drive

Upload the data, pretrianed model, figures, etc to your Google Drive, then mount this notebook to Google Drive. After that, you can access the resources freely.

Instruction: https://colab.research.google.com/notebooks/io.ipynb

Example: https://colab.research.google.com/drive/1srw_HFWQ2SMgmWIawucXfusGzrj1_U0q

Video: https://www.youtube.com/watch?v=zc8g8lGcwQU

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

[ ]:
```
import sys
import tensorflow as tf
print(sys.version)
print(tf.__version__)
```

4 Introduction

- Background of the problem
  - This study is about readmission/mortality prediction.
  - Unstructured data, especially for claim data which doesn't have clear structures, making models like MiME(Choi et al. 2018) not able to use.
  - The difficulties are finding the hidden structure of the data and making the prediction at the same time.
  - The paper's approach is effetive by their test metric.
- Paper explanation
  - The paper proposed a new approach, Graph Convolutional Transformer (GCT),to jointly lean the hidden strucuter and do the prediction task. This method will use the unstructured data as initial input and achieve good prediction of general medial tasks.
  - THE TEST METRIC FROM THE PAPER SHOWS BELOW
  - It is a great compliment for people who cannot access the structred data. Also the learned structure can be helpful for people who want reuse the learned sturcture for their furture study.

[ ]:
```
# code comment is used as inline annotations for your coding

img_dir = '/content/drive/My Drive/Colab_Notebooks/original_metircs.png'
current_dir = '/content/drive/My Drive/Colab_Notebooks'
import cv2
from google.colab.patches import cv2_imshow

img = cv2.imread(img_dir)

cv2_imshow(img)
```

5 Scope of Reproducibility:

List hypotheses from the paper you will test and the corresponding experiments you will run.

1. Hypothesis 1: The github repo(https://github.com/Google-Health/records-research/tree/master/graph-convolutional-transformer) as the paper shows can run and gives a meaningful result
2. Hypothesis 2: GCT outperforms baseline models in task readmission prediction for a publicly available EHR dataset

6 Methodology

This methodology is the core of your project. It consists of run-able codes with necessary annotations to show the expeiment you executed for testing the hypotheses.

The methodology at least contains two subsections **data** and **model** in your experiment.

7 Hypothesis 1: The github repo is runnable

With some fix, the repo is runnable based on the README.md.
Here are the steps I run the repo code:

1. fetch the repo into your local environment.
2. go to the directory of `graph-convolutional-transformer/`
3. Because of the compitable issue, remove the `import sklearn ...` in the two py files and change the function calls such as `ms.train_test_split` to `train_test_split`. Then add the function at the top of the two py files

   ```
   import random
   def train_test_split(data, test_size=0.2, random_state=None):
       if random_state is not None:
           random.seed(random_state)

       data_shuffled = data[:]
       random.shuffle(data_shuffled)

       split_idx = int(len(data) * (1 - test_size))

       train = data_shuffled[:split_idx]
       test = data_shuffled[split_idx:]

       return train, test
   ```

4. install conda and run below in terminal

   ```
   conda create -n DHLFinalProject python=2.7

   conda activate DHLFinalProject
   ```

```
conda install tensorflow==1.13.1
```

```
mkdir output
```

5. Copy data downloaded from citi to the directroy.
6. run below in terminal to process the eicu data.

   ```
   python eicu_samples/process_eicu.py . ./output
   ```

7. remove the `.` in the `train.py ./train.tfrecord`, `./validation.tfrecord`
8. create `trainResult/fold_0/` in the directory.
9. setting `num_iter = 4800`in `train.py`, then in terminal run: `python train.py output/fold_0 trainResult/fold_0`.
10. run for about 2hrs. We get the result in the termial as:

    ```
    INFO:tensorflow:Evaluation [10/100] INFO:tensorflow:Evaluation [20/100]
    INFO:tensorflow:Evaluation [30/100] INFO:tensorflow:Evaluation [40/100]
    INFO:tensorflow:Evaluation [50/100] INFO:tensorflow:Evaluation [60/100]
    INFO:tensorflow:Evaluation [70/100] INFO:tensorflow:Evaluation [80/100]
    INFO:tensorflow:Evaluation [90/100] INFO:tensorflow:Evaluation
    [100/100] INFO:tensorflow:Finished evaluation at 2024-04-13-09:18:05
    INFO:tensorflow:Saving dict for global step 4800: AUC-PR =
    0.32076856, AUC-ROC = 0.6733924, global_step = 4800, loss = 0.5876697
    INFO:tensorflow:Saving 'checkpoint_path' summary for global step
    4800: trainResult/fold_0/model.ckpt-4800 INFO:tensorflow:Loss
    for final step: 0.31709772. INFO:tensorflow:Calling model_fn.
    INFO:tensorflow:Done calling model_fn. INFO:tensorflow:Starting
    evaluation at 2024-04-13T09:18:06Z INFO:tensorflow:Graph
    was finalized. INFO:tensorflow:Restoring parameters from
    trainResult/fold_0/model.ckpt-4800 INFO:tensorflow:Running local_init_op.
    INFO:tensorflow:Done running local_init_op. 2024-04-13 02:18:07.447873: W
    ./tensorflow/core/grappler/optimizers/graph_optimizer_stage.h:241] Failed to
    run optimizer ArithmeticOptimizer, stage RemoveStackStridedSliceSameAxis
    node strided_slice_28. Error: ValidateStridedSliceOp returned partial
    shapes [1,?,1,101,101] and [?,1,101,101] 2024-04-13 02:18:07.447943: W
    ./tensorflow/core/grappler/optimizers/graph_optimizer_stage.h:241] Failed
    to run optimizer ArithmeticOptimizer, stage RemoveStackStridedSliceSameAxis
    node strided_slice_31. Error: ValidateStridedSliceOp returned partial
    shapes [1,?,1,101,101] and [?,1,101,101] 2024-04-13 02:18:07.447990: W
    ./tensorflow/core/grappler/optimizers/graph_optimizer_stage.h:241] Failed
    to run optimizer ArithmeticOptimizer, stage RemoveStackStridedSliceSameAxis
    node strided_slice_32. Error: ValidateStridedSliceOp returned partial shapes
    [1,?,1,101,101] and [?,1,101,101] INFO:tensorflow:Finished evaluation at
    2024-04-13-09:19:03 INFO:tensorflow:Saving dict for global step 4800: AUC-PR
    = 0.31366622, AUC-ROC = 0.676465, global_step = 4800, loss = 0.5836665
    INFO:tensorflow:Saving 'checkpoint_path' summary for global step 4800:
    trainResult/fold_0/model.ckpt-4800
    ```

8 Note:

1. The demo train reuslt in one fold is `AUC-PR = 0.31366622, AUC-ROC = 0.676465, global_step = 4800, loss = 0.5836665`.
2. The Hypothesis 1 can only be checked in local environment because the colab doesn't support the python2.7.
3. In my mac, 100 steps needs about 150 seconds to run in the train step. The model initially set the num_iter=1000000. This needs 35 days to run a five folds.
4. The **data** and **model** section is included above in a local environment.

9 Hypothesis 2: GCT outperforms baseline models in task readmission prediction for a publicly available EHR dataset

9.1 Data

Data includes raw data (MIMIC III tables), descriptive statistics (our homework questions), and data processing (feature engineering). * Source of the data: where the data is collected from; if data is synthetic or self-generated, explain how. If possible, please provide a link to the raw datasets. * Statistics: include basic descriptive statistics of the dataset like size, cross validation split, label distribution, etc. * Data process: how do you munipulate the data, e.g., change the class labels, split the dataset to train/valid/test, refining the dataset. * Illustration: printing results, plotting figures for illustration. * You can upload your raw dataset to Google Drive and mount this Colab to the same directory. If your raw dataset is too large, you can upload the processed dataset and have a code to load the processed dataset.

```
[ ]:
```

1. The eICU data comes from `https://physionet.org/content/eicu-crd/2.0/`. You are required to participate in the CITI training.
2. Download the patient, admissionDx, diagnosis, treatment CSV files
3. Decompress and upload to the Drive side by the notebook.

```
[ ]:
```

10 Notes

1. Because of the compitible isssues, it may be more approprate to rewrite the data process functions. But it takes more time than I thought. Currently It works but still have bugs.
2. Instead, I just fixed compitible isssues in the provided code and run the data process instead without using the pandas. I will leave pandas in future.

```
[ ]: # dir and function to load raw data
import pandas as pd

raw_data_dir = '/content/drive/My Drive/Colab_Notebooks/'
filenames = ['patient.csv', 'admissionDx.csv', 'diagnosis.csv', 'treatment.csv']
```

```
SUBSET_RATIO=0.1

def load_raw_data(raw_data_dir, filenames, subset_ratio=1.0):

    data_frames = []
    for filename in filenames:
        file_path = raw_data_dir + filename
        df = pd.read_csv(file_path)

        if subset_ratio < 1.0:
            df = df.sample(frac=subset_ratio)
        data_frames.append(df)

    return data_frames

raw_data_df_list = load_raw_data(raw_data_dir,filenames)
```

[ ]:
```
print('patient.csv')
raw_data_df_list[0].head(5)
```

[ ]:
```
print('admissionDx.csv')
raw_data_df_list[1].head(5)
```

[ ]:
```
print('diagnosis.csv')
raw_data_df_list[2].head(5)
```

[ ]:
```
print('treatment.csv')
raw_data_df_list[3].head(5)
```

[ ]:
```
# # process raw data
# def process_data(raw_data):
#     # implement this function to process the data as you need
#   return None

# processed_data = process_data(raw_data_df_list)

# ''' you can load the processed data directly
# processed_data_dir = '/content/gdrive/My Drive/Colab Notebooks/
 ↪<path-to-raw-data>'
# def load_processed_data(raw_data_dir):
#   pass

# '''

# calculate statistics
# include basic descriptive statistics of the dataset like size, cross␣
 ↪validation split, label distribution
```

```
def calculate_stats(raw_data,filenames):

    stats = [data.describe() for data in  raw_data]
    for filename,stat in zip(filenames,stats):
        print("Basic statistics for {}:\n {}".format(filename,stat))
        print('\n')

calculate_stats(raw_data_df_list, filenames)
```

[ ]:
```
# import pandas as pd

# class EncounterInfo(object):
#     def __init__(self, patient_id, encounter_id, encounter_timestamp,␣
↪expired, readmission):
#         self.patient_id = patient_id
#         self.encounter_id = encounter_id
#         self.encounter_timestamp = encounter_timestamp
#         self.expired = expired
#         self.readmission = readmission
#         self.dx_ids = []
#         self.rx_ids = []
#         self.labs = {}
#         self.physicals = []
#         self.treatments = []
#     def __repr__(self):
#         return (f"EncounterInfo(\n"
#                 f"  Patient ID: {self.patient_id},\n"
#                 f"  Encounter ID: {self.encounter_id},\n"
#                 f"  Timestamp: {self.encounter_timestamp},\n"
#                 f"  Expired: {'Yes' if self.expired else 'No'},\n"
#                 f"  Readmission: {'Yes' if self.readmission else 'No'},\n"
#                 f"  Diagnosis IDs: {self.dx_ids},\n"
#                 f"  Prescription IDs: {self.rx_ids},\n"
#                 f"  Labs: {self.labs},\n"
#                 f"  Physical Exams: {self.physicals},\n"
#                 f"  Treatments: {self.treatments}\n)")


# def process_patient(df, hour_threshold=24):
#     # Calculate encounter_timestamp and create a temporary DataFrame for␣
↪sorting
#     df['encounter_timestamp'] = -df['hospitaladmitoffset'].astype(int)

#     # Sorting patients by their IDs and then by the encounter timestamp
#     df_sorted = df.sort_values(['patienthealthsystemstayid',␣
↪'encounter_timestamp'])
```

```
#     # Detect readmissions by checking if the next stay is within the same␣
↪patient ID
#     df_sorted['next_patient_id'] = df_sorted['patienthealthsystemstayid'].
↪shift(-1)
#     df_sorted['readmission'] = df_sorted['patienthealthsystemstayid'] ==␣
↪df_sorted['next_patient_id']

#     # Mark the last encounter for each patient as not a readmission
#     df_sorted.loc[df_sorted['patienthealthsystemstayid'] !=␣
↪df_sorted['next_patient_id'], 'readmission'] = False

#     # Process each encounter
#     encounter_dict = {}
#     for _, row in df_sorted.iterrows():
#         duration_minute = float(row['unitdischargeoffset'])

#         if duration_minute > 60. * hour_threshold:
#             continue

#         expired = True if row['unitdischargestatus'] == 'Expired' else False
#         encounter_id = row['patientunitstayid']

#         # Instantiate EncounterInfo
#         ei = EncounterInfo(
#             patient_id=row['patienthealthsystemstayid'],
#             encounter_id=encounter_id,
#             encounter_timestamp=row['encounter_timestamp'],
#             expired=expired,
#             readmission=row['readmission']
#         )

#         if encounter_id in encounter_dict:
#             raise ValueError('Duplicate encounter ID!!')

#         encounter_dict[encounter_id] = ei

#     return encounter_dict

# encounter_dict = {}
# encounter_dict = process_patient(raw_data_df_list[0])
```

[ ]:
```
# print(next(iter(encounter_dict.items())))
```

[ ]:
```
# def process_admission_dx(df, encounter_dict):
#     # Check and report the number of missing encounter IDs
#     missing_eid = df[~df['patientunitstayid'].isin(encounter_dict.keys())]
#     print('Admission Diagnosis without Encounter ID:', len(missing_eid))
#     # Filter out rows where the encounter_id is not in encounter_dict
#     df = df[df['patientunitstayid'].isin(encounter_dict.keys())]

#     # Process each row in the DataFrame
#     for idx, row in df.iterrows():
#         encounter_id = row['patientunitstayid']
#         dx_id = row['admitdxpath'].lower()

#         if encounter_id in encounter_dict:
#             encounter_dict[encounter_id].dx_ids.append(dx_id)

#     return encounter_dict
# encounter_dict = process_admission_dx(raw_data_df_list[1],encounter_dict)
```

[ ]:
```
# print(next(iter(encounter_dict.items())))
```

[ ]:
```

```

[ ]:
```
# def process_diagnosis(df, encounter_dict):
#     # Count and report the number of missing encounter IDs
#     missing_eid = df[~df['patientunitstayid'].isin(encounter_dict.keys())]
#     print('Diagnosis without Encounter ID:', len(missing_eid))

#     # Filter the DataFrame to include only rows with encounter IDs that exist␣
 ↪in the encounter_dict
#     df = df[df['patientunitstayid'].isin(encounter_dict.keys())]

#     # Process each row to append diagnosis strings to the appropriate␣
 ↪EncounterInfo object
#     for _, row in df.iterrows():
#         encounter_id = row['patientunitstayid']
#         dx_id = row['diagnosisstring'].lower()  # Assuming case insensitivity␣
 ↪is desired

#         if encounter_id in encounter_dict:
#             encounter_dict[encounter_id].dx_ids.append(dx_id)

#     return encounter_dict
# encounter_dict = process_diagnosis(raw_data_df_list[2],encounter_dict)
```

[ ]:
```
# dict_iter = iter(encounter_dict.items())
# print(next(dict_iter))
# print(next(dict_iter))
```

```
[ ]: # def process_treatment(df, encounter_dict):
     #     # Identify and count missing encounter IDs that are not in the␣
      ↪encounter_dict
     #     missing_eid = df[~df['patientunitstayid'].isin(encounter_dict.keys())]
     #     print('Treatment without Encounter ID:', len(missing_eid))

     #     # Filter the DataFrame to include only rows with encounter IDs that exist␣
      ↪in the encounter_dict
     #     df_filtered = df[df['patientunitstayid'].isin(encounter_dict.keys())]

     #     # Process each row to append treatment strings to the appropriate␣
      ↪EncounterInfo object
     #     count = 0
     #     for _, row in df_filtered.iterrows():
     #         encounter_id = row['patientunitstayid']
     #         treatment_id = row['treatmentstring'].lower()  # Assuming case␣
      ↪insensitivity is desired

     #         if encounter_id in encounter_dict:
     #             encounter_dict[encounter_id].treatments.append(treatment_id)
     #             count += 1

     #     print('Accepted treatments:', count)
     #     return encounter_dict

     # encounter_dict = process_diagnosis(raw_data_df_list[3],encounter_dict)
```

```
[ ]: # dict_iter = iter(encounter_dict.items())
     # print(next(dict_iter))
     # print(next(dict_iter))
```

```
[ ]: # fix compitible issue of the github https://github.com/Alexuiuc/
      ↪records-research/tree/master/graph-convolutional-transformer:

     """Copyright 2019 Google LLC.

     Licensed under the Apache License, Version 2.0 (the "License");
     you may not use this file except in compliance with the License.
     You may obtain a copy of the License at

         https://www.apache.org/licenses/LICENSE-2.0

     Unless required by applicable law or agreed to in writing, software
     distributed under the License is distributed on an "AS IS" BASIS,
     WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
     See the License for the specific language governing permissions and
     limitations under the License.
```

```
"""

import pickle
import csv
import os
import sys
import numpy as np
import tensorflow as tf
import random

def train_test_split(data, test_size=0.2, random_state=None):
    if random_state is not None:
        random.seed(random_state)

    data_shuffled = data[:]
    random.shuffle(data_shuffled)

    split_idx = int(len(data) * (1 - test_size))

    train = data_shuffled[:split_idx]
    test = data_shuffled[split_idx:]

    return train, test

class EncounterInfo(object):

  def __init__(self, patient_id, encounter_id, encounter_timestamp, expired,
               readmission):
    self.patient_id = patient_id
    self.encounter_id = encounter_id
    self.encounter_timestamp = encounter_timestamp
    self.expired = expired
    self.readmission = readmission
    self.dx_ids = []
    self.rx_ids = []
    self.labs = {}
    self.physicals = []
    self.treatments = []


def process_patient(infile, encounter_dict, hour_threshold=24):

  inff = open(infile, 'r')

  count = 0
  patient_dict = {}
  for line in csv.DictReader(inff):
```

```
  patient_id = line['patienthealthsystemstayid']
  encounter_id = line['patientunitstayid']
  encounter_timestamp = -int(line['hospitaladmitoffset'])
  if patient_id not in patient_dict:
    patient_dict[patient_id] = []
  patient_dict[patient_id].append((encounter_timestamp, encounter_id))
inff.close()


patient_dict_sorted = {}
for patient_id, time_enc_tuples in patient_dict.items():
  patient_dict_sorted[patient_id] = sorted(time_enc_tuples)

enc_readmission_dict = {}
for patient_id, time_enc_tuples in patient_dict_sorted.items():
  for time_enc_tuple in time_enc_tuples[:-1]:
    enc_id = time_enc_tuple[1]
    enc_readmission_dict[enc_id] = True
  last_enc_id = time_enc_tuples[-1][1]
  enc_readmission_dict[last_enc_id] = False

inff = open(infile, 'r')
count = 0

for line in csv.DictReader(inff):

  patient_id = line['patienthealthsystemstayid']
  encounter_id = line['patientunitstayid']
  encounter_timestamp = -int(line['hospitaladmitoffset'])
  discharge_status = line['unitdischargestatus']
  duration_minute = float(line['unitdischargeoffset'])
  expired = True if discharge_status == 'Expired' else False
  readmission = enc_readmission_dict[encounter_id]

  if duration_minute > 60. * hour_threshold:
    continue

  ei = EncounterInfo(patient_id, encounter_id, encounter_timestamp, expired,
                     readmission)
  if encounter_id in encounter_dict:
    print('Duplicate encounter ID!!')
    sys.exit(0)
  encounter_dict[encounter_id] = ei
  count += 1

inff.close()
```

```python
  return encounter_dict


def process_admission_dx(infile, encounter_dict):
  inff = open(infile, 'r')
  count = 0
  missing_eid = 0
  for line in csv.DictReader(inff):

    encounter_id = line['patientunitstayid']
    dx_id = line['admitdxpath'].lower()

    if encounter_id not in encounter_dict:
      missing_eid += 1
      continue
    encounter_dict[encounter_id].dx_ids.append(dx_id)
    count += 1
  inff.close()
  print('')
  print('Admission Diagnosis without Encounter ID: %d' % missing_eid)

  return encounter_dict


def process_diagnosis(infile, encounter_dict):
  inff = open(infile, 'r')
  count = 0
  missing_eid = 0
  for line in csv.DictReader(inff):

    encounter_id = line['patientunitstayid']
    dx_id = line['diagnosisstring'].lower()

    if encounter_id not in encounter_dict:
      missing_eid += 1
      continue
    encounter_dict[encounter_id].dx_ids.append(dx_id)
    count += 1
  inff.close()
  print('')
  print('Diagnosis without Encounter ID: %d' % missing_eid)

  return encounter_dict
```

```python
def process_treatment(infile, encounter_dict):
  inff = open(infile, 'r')
  count = 0
  missing_eid = 0

  for line in csv.DictReader(inff):

    encounter_id = line['patientunitstayid']
    treatment_id = line['treatmentstring'].lower()

    if encounter_id not in encounter_dict:
      missing_eid += 1
      continue
    encounter_dict[encounter_id].treatments.append(treatment_id)
    count += 1
  inff.close()
  print('')
  print('Treatment without Encounter ID: %d' % missing_eid)
  print('Accepted treatments: %d' % count)

  return encounter_dict


def build_seqex(enc_dict,
                skip_duplicate=False,
                min_num_codes=1,
                max_num_codes=50):
  key_list = []
  seqex_list = []
  dx_str2int = {}
  treat_str2int = {}
  num_cut = 0
  num_duplicate = 0
  count = 0
  num_dx_ids = 0
  num_treatments = 0
  num_unique_dx_ids = 0
  num_unique_treatments = 0
  min_dx_cut = 0
  min_treatment_cut = 0
  max_dx_cut = 0
  max_treatment_cut = 0
  num_expired = 0
  num_readmission = 0

  for _, enc in enc_dict.items():
    if skip_duplicate:
      if (len(enc.dx_ids) > len(set(enc.dx_ids)) or
          len(enc.treatments) > len(set(enc.treatments))):
        num_duplicate += 1
        continue

    if len(set(enc.dx_ids)) < min_num_codes:
      min_dx_cut += 1
      continue

    if len(set(enc.treatments)) < min_num_codes:
      min_treatment_cut += 1
      continue

    if len(set(enc.dx_ids)) > max_num_codes:
      max_dx_cut += 1
      continue

    if len(set(enc.treatments)) > max_num_codes:
      max_treatment_cut += 1
      continue

    count += 1
    num_dx_ids += len(enc.dx_ids)
    num_treatments += len(enc.treatments)
    num_unique_dx_ids += len(set(enc.dx_ids))
    num_unique_treatments += len(set(enc.treatments))

    for dx_id in enc.dx_ids:
      if dx_id not in dx_str2int:
        dx_str2int[dx_id] = len(dx_str2int)

    for treat_id in enc.treatments:
      if treat_id not in treat_str2int:
        treat_str2int[treat_id] = len(treat_str2int)

    seqex = tf.train.SequenceExample()
    seqex.context.feature['patientId'].bytes_list.value.append((enc.patient_id +
                                                                 ':' +
                                                                 enc.
↪encounter_id).encode('utf-8'))
    if enc.expired:
      seqex.context.feature['label.expired'].int64_list.value.append(1)
      num_expired += 1
    else:
      seqex.context.feature['label.expired'].int64_list.value.append(0)
    if enc.readmission:
      seqex.context.feature['label.readmission'].int64_list.value.append(1)
      num_readmission += 1
    else:
      seqex.context.feature['label.readmission'].int64_list.value.append(0)

    dx_ids = seqex.feature_lists.feature_list['dx_ids']

    dx_ids_bytes = [dx_id.encode('utf-8') for dx_id in set(enc.dx_ids)]

    dx_ids.feature.add().bytes_list.value.extend(dx_ids_bytes)

    dx_int_list = [dx_str2int[item] for item in set(enc.dx_ids)]
    dx_ints = seqex.feature_lists.feature_list['dx_ints']
    dx_ints.feature.add().int64_list.value.extend(dx_int_list)


    proc_ids = seqex.feature_lists.feature_list['proc_ids']
    proc_ids_bytes = [item.encode('utf-8') for item in set(enc.treatments)]
    proc_ids.feature.add().bytes_list.value.extend(proc_ids_bytes)


    proc_int_list = [treat_str2int[item] for item in set(enc.treatments)]
    proc_ints = seqex.feature_lists.feature_list['proc_ints']
    proc_ints.feature.add().int64_list.value.extend(proc_int_list)


    seqex_list.append(seqex)
    key = seqex.context.feature['patientId'].bytes_list.value[0]  # assuming␣
↪this is set somewhere as bytes
    key_list.append(key)

  print('Filtered encounters due to duplicate codes: %d' % num_duplicate)
  print('Filtered encounters due to thresholding: %d' % num_cut)
  print('Average num_dx_ids: %f' % (num_dx_ids / count))
  print('Average num_treatments: %f' % (num_treatments / count))
  print('Average num_unique_dx_ids: %f' % (num_unique_dx_ids / count))
  print('Average num_unique_treatments: %f' % (num_unique_treatments / count))
  print('Min dx cut: %d' % min_dx_cut)
  print('Min treatment cut: %d' % min_treatment_cut)
  print('Max dx cut: %d' % max_dx_cut)
  print('Max treatment cut: %d' % max_treatment_cut)
  print('Number of expired: %d' % num_expired)
  print('Number of readmission: %d' % num_readmission)

  return key_list, seqex_list, dx_str2int, treat_str2int
```

```python
def select_train_valid_test(key_list, random_seed=1234):
  key_train, key_temp = train_test_split(
      key_list, test_size=0.2, random_state=random_seed)
  key_valid, key_test = train_test_split(
      key_temp, test_size=0.5, random_state=random_seed)
  return key_train, key_valid, key_test


def count_conditional_prob_dp(seqex_list, output_path, train_key_set=None):
  dx_freqs = {}
  proc_freqs = {}
  dp_freqs = {}
  total_visit = 0
  for seqex in seqex_list:
    if total_visit % 1000 == 0:
      sys.stdout.write('Visit count: %d\r' % total_visit)
      sys.stdout.flush()

    key = seqex.context.feature['patientId'].bytes_list.value[0]
    if (train_key_set is not None and key not in train_key_set):
      total_visit += 1
      continue

    dx_ids = seqex.feature_lists.feature_list['dx_ids'].feature[
        0].bytes_list.value
    proc_ids = seqex.feature_lists.feature_list['proc_ids'].feature[
        0].bytes_list.value

    for dx in dx_ids:
      if dx not in dx_freqs:
        dx_freqs[dx] = 0
      dx_freqs[dx] += 1

    for proc in proc_ids:
      if proc not in proc_freqs:
        proc_freqs[proc] = 0
      proc_freqs[proc] += 1

    for dx in dx_ids:
      for proc in proc_ids:
        dp = dx + b',' + proc
        if dp not in dp_freqs:
          dp_freqs[dp] = 0
        dp_freqs[dp] += 1

    total_visit += 1
  dx_probs = dict([(k, v / float(total_visit)) for k, v in dx_freqs.items()
                  ])
  proc_probs = dict([
      (k, v / float(total_visit)) for k, v in proc_freqs.items()
  ])
  dp_probs = dict([(k, v / float(total_visit)) for k, v in dp_freqs.items()
                  ])

  dp_cond_probs = {}
  pd_cond_probs = {}
  for dx, dx_prob in dx_probs.items():
    for proc, proc_prob in proc_probs.items():
      dp = dx + b',' + proc
      pd = proc + b',' + dx
      if dp in dp_probs:
        dp_cond_probs[dp] = dp_probs[dp] / dx_prob
        pd_cond_probs[pd] = dp_probs[dp] / proc_prob
      else:
        dp_cond_probs[dp] = 0.0
        pd_cond_probs[pd] = 0.0

  pickle.dump(dx_probs, open(output_path + '/dx_probs.empirical.p', 'wb'), -1)
  pickle.dump(proc_probs, open(output_path + '/proc_probs.empirical.p', 'wb'),
              -1)
  pickle.dump(dp_probs, open(output_path + '/dp_probs.empirical.p', 'wb'), -1)
  pickle.dump(dp_cond_probs,
              open(output_path + '/dp_cond_probs.empirical.p', 'wb'), -1)
  pickle.dump(pd_cond_probs,
              open(output_path + '/pd_cond_probs.empirical.p', 'wb'), -1)


def add_sparse_prior_guide_dp(seqex_list,
                              stats_path,
                              key_set=None,
                              max_num_codes=50):
  print('Loading conditional probabilities.')
  dp_cond_probs = pickle.load(
      open(stats_path + '/dp_cond_probs.empirical.p', 'rb'))
  pd_cond_probs = pickle.load(
      open(stats_path + '/pd_cond_probs.empirical.p', 'rb'))

  print('Adding prior guide.')
  total_visit = 0
  new_seqex_list = []
  for seqex in seqex_list:
    if total_visit % 1000 == 0:
      sys.stdout.write('Visit count: %d\r' % total_visit)
      sys.stdout.flush()

    key = seqex.context.feature['patientId'].bytes_list.value[0]
    if (key_set is not None and key not in key_set):
      total_visit += 1
      continue

    dx_ids = seqex.feature_lists.feature_list['dx_ids'].feature[
        0].bytes_list.value
    proc_ids = seqex.feature_lists.feature_list['proc_ids'].feature[
        0].bytes_list.value

    indices = []
    values = []
    for i, dx in enumerate(dx_ids):
      for j, proc in enumerate(proc_ids):
        dp = dx + b',' + proc
        indices.append((i, max_num_codes + j))
        prob = 0.0 if dp not in dp_cond_probs else dp_cond_probs[dp]
        values.append(prob)

    for i, proc in enumerate(proc_ids):
      for j, dx in enumerate(dx_ids):
        pd = proc + b',' + dx
        indices.append((max_num_codes + i, j))
        prob = 0.0 if pd not in pd_cond_probs else pd_cond_probs[pd]
        values.append(prob)

    indices = list(np.array(indices).reshape([-1]))
    indices_feature = seqex.feature_lists.feature_list['prior_indices']
    indices_feature.feature.add().int64_list.value.extend(indices)
    values_feature = seqex.feature_lists.feature_list['prior_values']
    values_feature.feature.add().float_list.value.extend(values)

    new_seqex_list.append(seqex)
    total_visit += 1

  return new_seqex_list


"""Set <input_path> to where the raw eICU CSV files are located.
Set <output_path> to where you want the output files to be.
"""
def data_process():
  input_path = current_dir
  output_path = current_dir+'/output'
```

```
num_fold = 3

patient_file = input_path + '/patient.csv'
admission_dx_file = input_path + '/admissionDx.csv'
diagnosis_file = input_path + '/diagnosis.csv'
treatment_file = input_path + '/treatment.csv'

encounter_dict = {}
print('Processing patient.csv')
encounter_dict = process_patient(
    patient_file, encounter_dict, hour_threshold=24)
print('Processing admission diagnosis.csv')
encounter_dict = process_admission_dx(admission_dx_file, encounter_dict)
print('Processing diagnosis.csv')
encounter_dict = process_diagnosis(diagnosis_file, encounter_dict)
print('Processing treatment.csv')
encounter_dict = process_treatment(treatment_file, encounter_dict)

key_list, seqex_list, dx_map, proc_map = build_seqex(
    encounter_dict, skip_duplicate=False, min_num_codes=1, max_num_codes=50)

pickle.dump(dx_map, open(output_path + '/dx_map.p', 'wb'), -1)
pickle.dump(proc_map, open(output_path + '/proc_map.p', 'wb'), -1)

for i in range(num_fold):
  fold_path = output_path + '/fold_' + str(i)
  stats_path = fold_path + '/train_stats'
  os.makedirs(stats_path)

  key_train, key_valid, key_test = select_train_valid_test(
      key_list, random_seed=i)

  count_conditional_prob_dp(seqex_list, stats_path, set(key_train))
  train_seqex = add_sparse_prior_guide_dp(
      seqex_list, stats_path, set(key_train), max_num_codes=50)
  validation_seqex = add_sparse_prior_guide_dp(
      seqex_list, stats_path, set(key_valid), max_num_codes=50)
  test_seqex = add_sparse_prior_guide_dp(
      seqex_list, stats_path, set(key_test), max_num_codes=50)

  with tf.io.TFRecordWriter(fold_path + '/train.tfrecord') as writer:
    for seqex in train_seqex:
      writer.write(seqex.SerializeToString())

  with tf.io.TFRecordWriter(fold_path + '/validation.tfrecord') as writer:
    for seqex in validation_seqex:
      writer.write(seqex.SerializeToString())
    with tf.io.TFRecordWriter(fold_path + '/test.tfrecord') as writer:
      for seqex in test_seqex:
        writer.write(seqex.SerializeToString())


%time data_process()
```

10.1 Model

The model includes the model definitation which usually is a class, model training, and other necessary parts. * Model architecture: layer number/size/type, activation function, etc * Training objectives: loss function, optimizer, weight of each loss term, etc * Others: whether the model is pretrained, Monte Carlo simulation for uncertainty analysis, etc * The code of model should have classes of the model, functions of model training, model validation, etc. * If your model training is done outside of this notebook, please upload the trained model here and develop a function to load and test it.

11 Note

1. I attempted to convert the model to a version compatible with TensorFlow 2.0 and Python 3, but encountered internal errors that made training impossible.
2. One possible reason is that the Estimator used by TensorFlow is quite outdated, and I may have mixed some API usages.
3. The next steps are:

- Try to debug line by line, because it can run on my local machine's Python 2.7.
- Convert the model to PyTorch. I have reviewed the model's 2,000 lines of code.

[ ]:
```
# class my_model():
#   # use this class to define your model
#   pass

# model = my_model()
# loss_func = None
# optimizer = None

# def train_model_one_iter(model, loss_func, optimizer):
#   pass

# num_epoch = 10
# # model training loop: it is better to print the training/validation losses␣
 ↪during the training
# for i in range(num_epoch):
#   train_model_one_iter(model, loss_func, optimizer)
#   train_loss, valid_loss = None, None
#   print("Train Loss: %.2f, Validation Loss: %.2f" % (train_loss, valid_loss))
```

```python
"""Copyright 2019 Google LLC.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
"""

import tensorflow as tf
import sys


class FeatureEmbedder(object):
    def __init__(self, vocab_sizes, feature_keys, embedding_size):
        """Initialize the feature embedder."""
        self._params = {}
        self._feature_keys = feature_keys
        self._vocab_sizes = vocab_sizes
        dummy_emb = tf.zeros([1, embedding_size], dtype=tf.float32)

        for feature_key in feature_keys:
            vocab_size = self._vocab_sizes[feature_key]
            emb = tf.Variable(tf.random.uniform([vocab_size, embedding_size]),␣
 ↪name=feature_key)
            self._params[feature_key] = tf.concat([emb, dummy_emb], axis=0)

        self._params['visit'] = tf.Variable(tf.random.uniform([1,␣
 ↪embedding_size]), name='visit')

    def lookup(self, feature_map, max_num_codes):
        """Convert SparseTensors to dense embeddings and masks."""
        masks = {}
        embeddings = {}
        for key in self._feature_keys:
            if max_num_codes > 0:
                feature = tf.SparseTensor(
                indices=feature_map[key].indices,
                values=feature_map[key].values,
                dense_shape=[
                    feature_map[key].dense_shape[0],
                    feature_map[key].dense_shape[1], max_num_codes
                ])
            else:
                feature = feature_map[key]
            feature_ids = tf.sparse.to_dense(
              feature, default_value=self._vocab_sizes[key])
            feature_ids = tf.squeeze(feature_ids, axis=1)
            embeddings[key] = tf.nn.embedding_lookup(self._params[key],␣
 ↪feature_ids)

            mask = tf.SparseTensor(
              indices=feature.indices,
              values=tf.ones(tf.shape(feature.values)),
              dense_shape=feature.dense_shape)
            masks[key] = tf.squeeze(tf.sparse.to_dense(mask), axis=1)

        batch_size = tf.shape(list(embeddings.values())[0])[0]
        embeddings['visit'] = tf.tile(self._params['visit'][None, :, :],
                                      [batch_size, 1, 1])

        masks['visit'] = tf.ones(batch_size)[:, None]

        return embeddings, masks



class GraphConvolutionalTransformer(tf.keras.layers.Layer):
    """Init function.

    Args:
      embedding_size: The size of the dimension for hidden layers.
      num_transformer_stack: The number of Transformer blocks.
      num_feedforward: The number of layers in the feedforward part of
        Transformer.
      num_attention_heads: The number of attention heads.
      ffn_dropout: Dropout rate used inside the feedforward part.
      attention_normalizer: Use either 'softmax' or 'sigmoid' to normalize the
        attention values.
      multihead_attention_aggregation: Use either 'concat' or 'sum' to handle
        the outputs from multiple attention heads.
      directed_attention: Decide whether you want to use the unidirectional
        attention, where information accumulates inside the dummy visit node.
      use_inf_mask: Decide whether you want to use the guide matrix. Currently
        unused.
      use_prior: Decide whether you want to use the conditional probablility
        information. Currently unused.
      **kwargs: Other arguments to tf.keras.layers.Layer init.
    """
    def __init__(self,
               embedding_size=128,
               num_transformer_stack=3,
               num_feedforward=2,
               num_attention_heads=1,
               ffn_dropout=0.1,
               attention_normalizer='softmax',
               multihead_attention_aggregation='concat',
               directed_attention=False,
               use_inf_mask=True,
               use_prior=True,
               **kwargs):

        super(GraphConvolutionalTransformer, self).__init__(**kwargs)
        self._hidden_size = embedding_size
        self._num_stack = num_transformer_stack
        self._num_feedforward = num_feedforward
        self._num_heads = num_attention_heads
        self._ffn_dropout = ffn_dropout
        self._attention_normalizer = attention_normalizer
        self._multihead_aggregation = multihead_attention_aggregation
        self._directed_attention = directed_attention
        self._use_inf_mask = use_inf_mask
        self._use_prior = use_prior

        self._layers = {}
        self._layers['Q'] = []
        self._layers['K'] = []
        self._layers['V'] = []
        self._layers['ffn'] = []
        self._layers['head_agg'] = []
        self.layer_norm = tf.keras.layers.LayerNormalization(axis=2)

        for i in range(self._num_stack):
            self._layers['Q'].append(
              tf.keras.layers.Dense(
                  self._hidden_size * self._num_heads, use_bias=False))
            self._layers['K'].append(
              tf.keras.layers.Dense(
                  self._hidden_size * self._num_heads, use_bias=False))
            self._layers['V'].append(
              tf.keras.layers.Dense(
                  self._hidden_size * self._num_heads, use_bias=False))

            if self._multihead_aggregation == 'concat':
                self._layers['head_agg'].append(
                tf.keras.layers.Dense(self._hidden_size, use_bias=False))

            self._layers['ffn'].append([])
          # Don't need relu for the last feedforward.
            for _ in range(self._num_feedforward - 1):
                self._layers['ffn'][i].append(
                tf.keras.layers.Dense(self._hidden_size, activation='relu'))
            self._layers['ffn'][i].append(tf.keras.layers.Dense(self.
↪_hidden_size))

    def feedforward(self, features, stack_index, training=None):
        """Feedforward component of Transformer.

        Args:
          features: 3D float Tensor of size (batch_size, num_features,
            embedding_size). This is the input embedding to GCT.
          stack_index: An integer to indicate which Transformer block we are in.
          training: Whether to run in training or eval mode.

        Returns:
          Latent representations derived from this feedforward network.
        """
        for i in range(self._num_feedforward):
            features = self._layers['ffn'][stack_index][i](features)
            if training:
                features = tf.nn.dropout(features, rate=self._ffn_dropout)

        return features

    def qk_op(self,
            features,
            stack_index,
            batch_size,
            num_codes,
            attention_mask,
            inf_mask=None,
            directed_mask=None):
        """Attention generation part of Transformer.

        Args:
          features: 3D float Tensor of size (batch_size, num_features,
            embedding_size). This is the input embedding to GCT.
          stack_index: An integer to indicate which Transformer block we are in.
          batch_size: The size of the mini batch.
          num_codes: The number of features (i.e. codes) given as input.
          attention_mask: A Tensor for suppressing the attention on the padded
            tokens.
          inf_mask: The guide matrix to suppress the attention values to zeros␣
  ↪for
            certain parts of the attention matrix (e.g. diagnosis codes cannot
            attend to other diagnosis codes).
          directed_mask: If the user wants to only use the upper-triangle of the
            attention for uni-directional attention flow, we use this strictly␣
 ↪lower
            triangular matrix filled with infinity.

        Returns:
          The attention distribution derived from the QK operation.
        """

        q = self._layers['Q'][stack_index](features)
        q = tf.reshape(q,
                       [batch_size, num_codes, self._hidden_size, self.
 ↪_num_heads])

        k = self._layers['K'][stack_index](features)
        k = tf.reshape(k,
                       [batch_size, num_codes, self._hidden_size, self.
 ↪_num_heads])

        # Need to transpose q and k to (2, 0, 1)
        q = tf.transpose(q, perm=[0, 3, 1, 2])
        k = tf.transpose(k, perm=[0, 3, 2, 1])
        pre_softmax = tf.matmul(q, k) / tf.sqrt(
            tf.cast(self._hidden_size, tf.float32))

        pre_softmax -= attention_mask[:, None, None, :]

        if inf_mask is not None:
            pre_softmax -= inf_mask[:, None, :, :]

        if directed_mask is not None:
            pre_softmax -= directed_mask

        if self._attention_normalizer == 'softmax':
            attention = tf.nn.softmax(pre_softmax, axis=3)
        else:
            attention = tf.nn.sigmoid(pre_softmax)
        return attention

    def call(self, features,masks, guide=None, prior_guide=None, training=None):
        """Transforms input embeddings to output embeddings with attention␣
 ↪mechanism."""
        batch_size = tf.shape(features)[0]
```

```
        num_codes = tf.shape(features)[1]

        # Use masks to create an attention mask
        mask_idx = tf.cast(tf.where(tf.equal(masks[:, :, 0], 0)), tf.int32)
        attention_mask = tf.fill(tf.shape(features)[:-1], -1e9)  # Use large␣
↪negative to simulate -inf
        attention_mask = tf.tensor_scatter_nd_update(attention_mask, mask_idx,␣
↪tf.zeros_like(mask_idx[:, 0], dtype=tf.float32))

        inf_mask = None
        if self._use_inf_mask:
            inf_mask = tf.cast(tf.equal(guide, 0), tf.float32) * -1e9

        directed_mask = None
        if self._directed_attention:
            lower_triangular = tf.linalg.band_part(tf.ones((num_codes,␣
↪num_codes)), -1, 0)
            directed_mask = -1e9 * (1.0 - lower_triangular)

        attentions = []
        for i in range(self._num_stack):
            if self._use_prior and i == 0:
                attention_scores = tf.tile(prior_guide[:, None, :, :], [1, self.
↪_num_heads, 1, 1])
            else:
                attention_scores = self.qk_op(features, i, batch_size,␣
↪num_codes, attention_mask, inf_mask, directed_mask)

            attentions.append(attention_scores)

            v = self._layers['V'][i](features)
            v = tf.reshape(v, [batch_size, num_codes, self._hidden_size, self.
↪_num_heads])
            v = tf.transpose(v, perm=[0, 3, 1, 2])
            post_attention = tf.matmul(attention_scores, v)

            if self._num_heads == 1:
                post_attention = tf.squeeze(post_attention, axis=1)
            elif self._multihead_aggregation == 'concat':
                post_attention = tf.transpose(post_attention, perm=[0, 2, 1, 3])
                post_attention = tf.reshape(post_attention, [batch_size,␣
↪num_codes, -1])
                post_attention = self._layers['head_agg'][i](post_attention)
            else:
                post_attention = tf.reduce_sum(post_attention, axis=1)
            # Residual connection and layer normalization
            post_attention += features
            post_attention = self.layer_norm(post_attention)  # Apply␣
↪LayerNormalization

            # Feedforward network
            post_ffn = self.feedforward(post_attention, i, training)
            post_ffn += post_attention
            post_ffn = self.layer_norm(post_ffn)  # Apply LayerNormalization␣
↪again

            features = post_ffn

        return features * masks, attentions



def create_matrix_vdp(features, mask, use_prior, use_inf_mask, max_num_codes,␣
↪prior_scalar):
    dx_ids = features['dx_ints']
    proc_ids = features['proc_ints']

    batch_size = dx_ids.dense_shape[0]
    num_dx_ids = max_num_codes if use_prior else dx_ids.dense_shape[-1]
    num_proc_ids = max_num_codes if use_prior else proc_ids.dense_shape[-1]
    num_codes = 1 + num_dx_ids + num_proc_ids

    guide = None
    if use_inf_mask:
        row0 = tf.concat([
            tf.zeros([1, 1]),
            tf.ones([1, num_dx_ids]),
            tf.zeros([1, num_proc_ids])
        ], axis=1)

        row1 = tf.concat([
            tf.zeros([num_dx_ids, 1 + num_dx_ids]),
            tf.ones([num_dx_ids, num_proc_ids])
        ], axis=1)

        row2 = tf.zeros([num_proc_ids, num_codes])

        guide = tf.concat([row0, row1, row2], axis=0)
        guide = guide + tf.transpose(guide)
        guide = tf.tile(guide[None, :, :], [batch_size, 1, 1])
        guide *= mask[:, :, None] * mask[:, None, :]
        guide += tf.eye(num_codes)[None, :, :]
    prior_guide = None
    if use_prior:
        prior_values = features['prior_values']
        prior_indices = features['prior_indices']
        prior_batch_idx = prior_indices.indices[:, 0][::2]
        prior_idx = tf.reshape(prior_indices.values, [-1, 2])
        prior_idx = tf.concat(
            [prior_batch_idx[:, None], prior_idx[:, :1], prior_idx[:, 1:]],␣
↪axis=1)

        temp_idx = (
            prior_idx[:, 0] * 1000000 + prior_idx[:, 1] * 1000 + prior_idx[:,␣
↪2])
        sorted_idx = tf.argsort(temp_idx)
        prior_idx = tf.gather(prior_idx, sorted_idx)

        prior_idx_shape = [batch_size, max_num_codes * 2, max_num_codes * 2]
        sparse_prior = tf.SparseTensor(
            indices=prior_idx, values=prior_values.values,␣
↪dense_shape=prior_idx_shape)
        prior_guide = tf.sparse.to_dense(sparse_prior, default_value=0)

        visit_guide = tf.convert_to_tensor(
            [prior_scalar] * max_num_codes + [0.0] * max_num_codes,
            dtype=tf.float32)
        prior_guide = tf.concat(
            [tf.tile(visit_guide[None, None, :], [batch_size, 1, 1]),␣
↪prior_guide],
            axis=1)
        visit_guide = tf.concat([[0.0], visit_guide], axis=0)
        prior_guide = tf.concat(
            [tf.tile(visit_guide[None, :, None], [batch_size, 1, 1]),␣
↪prior_guide],
            axis=2)
        prior_guide *= mask[:, :, None] * mask[:, None, :]
        prior_guide += prior_scalar * tf.eye(num_codes)[None, :, :]
        degrees = tf.reduce_sum(prior_guide, axis=2)
        prior_guide /= degrees[:, :, None]

    return guide, prior_guide


class SequenceExampleParser(object):
    def __init__(self, batch_size, num_map_threads=4):
        self.context_features_config = {
            'patientId': tf.io.VarLenFeature(dtype=tf.string),
            'label.readmission': tf.io.FixedLenFeature(shape=[1], dtype=tf.
↪int64),
            'label.expired': tf.io.FixedLenFeature(shape=[1], dtype=tf.int64)
        }
        self.sequence_features_config = {
            'dx_ints': tf.io.VarLenFeature(dtype=tf.int64),
            'proc_ints': tf.io.VarLenFeature(dtype=tf.int64),
            'prior_indices': tf.io.VarLenFeature(dtype=tf.int64),
            'prior_values': tf.io.VarLenFeature(dtype=tf.float32)
        }
        self.batch_size = batch_size
        self.num_map_threads = num_map_threads

    def __call__(self, tfrecord_path, label_key, training):
        def parser_fn(serialized_example):
            (batch_context, batch_sequence) = tf.io.
↪parse_single_sequence_example(
                serialized_example,
                context_features=self.context_features_config,
                sequence_features=self.sequence_features_config)
            labels = tf.squeeze(tf.cast(batch_context[label_key], tf.float32))
            return batch_sequence, labels

        num_epochs = None if training else 1
        buffer_size = self.batch_size * 32
        dataset = tf.data.TFRecordDataset(tfrecord_path)
        dataset = dataset.shuffle(buffer_size) if training else dataset
        dataset = dataset.repeat(num_epochs)
        dataset = dataset.map(parser_fn, num_parallel_calls=self.
↪num_map_threads)
        dataset = dataset.batch(self.batch_size)
        dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)

        return dataset



class EHRTransformer(object):
    """Transformer-based EHR encounter modeling algorithm.

    All features within each encounter are put through multiple steps of
    self-attention. There is a dummy visit embedding in addition to other
    feature embeddings, which can be used for encounter-level predictions.
    """

    def __init__(self,
               gct_params,
               feature_keys=['dx_ints', 'proc_ints'],
               label_key='label.readmission',
               vocab_sizes={'dx_ints':3249, 'proc_ints':2210},
               feature_set='vdp',
               max_num_codes=50,
               prior_scalar=0.5,
               reg_coef=0.1,
               num_classes=1,
               learning_rate=1e-3,
               batch_size=32):
        """Init function.

        Args:
          gct_params: A dictionary parameteres to be used inside GCT class. See␣
↪GCT
            comments for more information.
          feature_keys: A list of feature names you want to use. (e.g.␣
 ↪['dx_ints,
            'proc_ints', 'lab_ints'])
          vocab_sizes: A dictionary of vocabularize sizes for each feature. (e.
 ↪g.
            {'dx_ints': 1001, 'proc_ints': 1001, 'lab_ints': 1001})
          feature_set: Use 'vdpl' to indicate your features are diagnosis codes,
            treatment codes, and lab codes. Use 'vdp' to indicate your features␣
 ↪are
            diagnosis codes and treatment codes.
          max_num_codes: The maximum number of how many feature there can be␣
 ↪inside
            a single visit, per feature. For example, if this is set to 50,␣
 ↪then we
            are assuming there can be up to 50 diagnosis codes, 50 treatment␣
 ↪codes,
            and 50 lab codes. This will be used for creating the prior matrix.
          prior_scalar: A float value between 0.0 and 1.0 to be used to␣
 ↪hard-code
            the diagnoal elements of the prior matrix.
          reg_coef: A coefficient to decide the KL regularization balance when
            training GCT.
          num_classes: This is set to 1, because this implementation only␣
 ↪supports
            graph-level binary classification.
          learning_rate: Learning rate for Adam optimizer.
          batch_size: Batch size.
        """
        self._feature_keys = feature_keys
        self._label_key = label_key
        self._vocab_sizes = vocab_sizes
        self._feature_set = feature_set
        self._max_num_codes = max_num_codes
        self._prior_scalar = prior_scalar
        self._reg_coef = reg_coef
        self._num_classes = num_classes
        self._learning_rate = learning_rate
        self._batch_size = batch_size

        self._gct_params = gct_params
        self._embedding_size = gct_params['embedding_size']
        self._num_transformer_stack = gct_params['num_transformer_stack']
        self._use_inf_mask = gct_params['use_inf_mask']
        self._use_prior = gct_params['use_prior']

        self._seqex_reader = SequenceExampleParser(self._batch_size)

        self.dense_layer = tf.keras.layers.Dense(self._num_classes,␣
↪activation=None)

        self.loss_function = tf.keras.losses.
↪BinaryCrossentropy(from_logits=True)

    def get_prediction(self, model, feature_embedder, features, training=False):

        embedding_dict, mask_dict = feature_embedder.lookup(features, self.
↪_max_num_codes)
        keys = ['visit'] + self._feature_keys
        embeddings = tf.concat([embedding_dict[key] for key in keys], axis=1)

        #print([mask_dict[key].shape for key in keys])

        masks = tf.concat([mask_dict[key] for key in keys], axis=1)

        # Assuming create_matrix_vdp is updated for TensorFlow 2.x
        guide, prior_guide = create_matrix_vdp(features, masks, self._use_prior,
                                               self._use_inf_mask, self.
↪_max_num_codes,
                                               self._prior_scalar)

        hidden, attentions = model(embeddings, masks[:, :, None], guide,␣
↪prior_guide, training)

        pre_logit = hidden[:, 0, :]
        pre_logit = tf.reshape(pre_logit, [-1, self._embedding_size])
        logits = self.dense_layer(pre_logit)
        logits = tf.squeeze(logits)

        return logits, attentions

    def get_loss(self, logits, labels, attentions):
        loss = self.loss_function(labels, logits)

        if self._use_prior:
            kl_terms = []
            attention_tensor = tf.stack(attentions)
            for i in range(1, self._num_transformer_stack):
                log_p = tf.math.log(attention_tensor[i - 1] + 1e-12)
                log_q = tf.math.log(attention_tensor[i] + 1e-12)
                kl_term = attention_tensor[i - 1] * (log_p - log_q)
                kl_term = tf.reduce_sum(kl_term, axis=-1)  # Sum across the␣
↪last dimension
                kl_term = tf.reduce_mean(kl_term)  # Mean across the batch
                kl_terms.append(kl_term)

            reg_term = tf.reduce_mean(kl_terms)  # Mean across all layers
            loss += self._reg_coef * reg_term

        return loss

    def input_fn(self, tfrecord_path, training):
        """Input function to be used by TensorFlow Estimator.

        Args:
          tfrecord_path: Path to TFRecord of SequenceExamples.
          training: Boolean value to indicate whether the model if training.

        Return:
          Input generator.
        """
        return self._seqex_reader(tfrecord_path, self._label_key, training)


    def model_fn(self, features, labels, mode, params):
        training = mode == tf.estimator.ModeKeys.TRAIN

        # Initialize model and feature embedder
        model = GraphConvolutionalTransformer(**self._gct_params)
        feature_embedder = FeatureEmbedder(
            self._vocab_sizes, self._feature_keys, self._embedding_size)

        logits, attentions = self.get_prediction(model, feature_embedder,␣
↪features, training)
```

```
        probs = tf.nn.sigmoid(logits)

        predictions = {
            'probabilities': probs,
            'logits': logits,
        }

        # Output predictions in PREDICT mode
        if mode == tf.estimator.ModeKeys.PREDICT:
            return tf.estimator.EstimatorSpec(mode, predictions=predictions)

        # Compute loss
        loss = get_loss(logits, labels, attentions)

        if mode == tf.estimator.ModeKeys.TRAIN:
            optimizer = tf.keras.optimizers.Adam(learning_rate=self.
↪_learning_rate)
            train_op = optimizer.minimize(loss, tf.compat.v1.train.
↪get_or_create_global_step())
            return tf.estimator.EstimatorSpec(mode, loss=loss,␣
↪train_op=train_op)

        elif mode == tf.estimator.ModeKeys.EVAL:
            # Define the metrics
            metrics_dict = {
                'AUC-PR': tf.keras.metrics.AUC(labels, probs, curve='PR'),
                'AUC-ROC': tf.keras.metrics.AUC(labels, probs, curve='ROC')
            }
            return tf.estimator.EstimatorSpec(mode, loss=loss,␣
↪eval_metric_ops=metrics_dict)
```

[ ]:

```
import tensorflow as tf
import logging

def main():
    gct_params = {
      "embedding_size": 128,
      "num_transformer_stack": 3,
      "num_feedforward": 2,
      "num_attention_heads": 1,
      "ffn_dropout": 0.08,
      "attention_normalizer": "softmax",
      "multihead_attention_aggregation": "concat",
      "directed_attention": False,
      "use_inf_mask": True,
      "use_prior": True,
    }

    input_path = '/output/fold_0'
    model_dir = current_dir+'/result'
    #   num_iter = 1000000
    num_iter = 4800  # 2 hours for 100 per 115 second in my mac.
    model = EHRTransformer(
      gct_params=gct_params,
      label_key='label.readmission',
      reg_coef=0.1,
      learning_rate=0.00022,
      batch_size=32)
    config = tf.estimator.RunConfig(save_checkpoints_steps=100)

    estimator = tf.estimator.Estimator(
      model_dir=model_dir, model_fn=model.model_fn, config=config)

    train_spec = tf.estimator.TrainSpec(
      input_fn=lambda: model.input_fn(input_path + '/train.tfrecord', True),
      max_steps=num_iter)

    eval_spec = tf.estimator.EvalSpec(
      input_fn=lambda: model.input_fn(input_path + '/validation.tfrecord',␣
 ↪False),
      throttle_secs=1)

    tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)

    estimator.evaluate(
      input_fn=lambda: model.input_fn(input_path + '/validation.tfrecord',␣
 ↪False))
```

```
logging.getLogger('tensorflow').setLevel(logging.INFO)
if __name__ == '__main__':

    main()
```

12 Results

In this section, you should finish training your model training or loading your trained model. That is a great experiment! You should share the results with others with necessary metrics and figures.

Please test and report results for all experiments that you run with:

- specific numbers (accuracy, AUC, RMSE, etc)
- figures (loss shrinkage, outputs from GAN, annotation or label of sample pictures, etc)

[ ]:
```
# metrics to evaluate my model
print('run in my local python2.7 and test by using the test file, It gives the␣
 ↪following result:')
print("Test results: {'AUC-ROC': 0.5, 'AUC-PR': 0.16597612, 'global_step':␣
 ↪1000, 'loss': 0.6367319}")
# plot figures to better show the results

# it is better to save the numbers and figures for your presentation.
```

12.1 Model comparison

[ ]:
```
# compare you model with others
# you don't need to re-run all other experiments, instead, you can directly␣
 ↪refer the metrics/numbers in the paper
print("The above result 0.16597612 is much worse than the paper's result 0.5965␣
 ↪and all other models")
print("This is due to the less training time and other selected parameter like␣
 ↪learning rate")
```

13 Discussion

In this section,you should discuss your work and make future plan. The discussion should address the following questions: * Make assessment that the paper is reproducible or not. * Explain why it is not reproducible if your results are kind negative. * Describe "What was easy" and "What was difficult" during the reproduction. * Make suggestions to the author or other reproducers on how to improve the reproducibility. * What will you do in next phase.

[ ]:
```
# no code is required for this section
'''
if you want to use an image outside this notebook for explanaition,
you can read and plot it here like the Scope of Reproducibility
```

```
'''
```

14 References

1. Learning the Graphical Structure of Electronic Health Records with Graph Convolutional Transformer Edward Choi, Zhen Xu, Yujia Li, Michael W. Dusenberry, Gerardo Flores, Yuan Xue, Andrew M. Dai
AAAI 2020

15 Feel free to add new sections