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.

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.

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')
```

```
Mounted at /content/drive
```

Introduction

Our team referenced research paper "A machine learning approach to identifying delirium from electronic health records" to analyze the detection of delirium.

The identication of it has always been difficult due to inadepquate assessment and under-documentation. Many of the cases are identified after a period of meidication usage or ICU admission. The focus of our chosen paper is to present a classification model that identifies delirium using retrospective EHR data. The goal of our team is to understand the problem and method introduced by the paper in order to replicate it to achieve similar results based on MIMIC III health datasets. We want to further prove the point that through logistic regression model and multi-layer perception can demosntrate a high accuracy with a mean AU of 0.87.

```
# code comment is used as inline annotations for your coding
!ls "/content/drive/My Drive/Colab Notebooks"
```

```
demo_records.pkl  labels.csv  labels.pkl  master.csv  ml_output  patient_records.pkl
```

Double-click (or enter) to edit

Scope of Reproducibility:

The main hypothesis we are going to test will align with the research paper:

Machine learning models can identify delirium from electronic health record data with greater accuracy than traditional screening methods.

The project duration is estimated to be just a little over a month, the initial phase will focus on the data retrieve and processing to ensure we have access to all neccessary criteria/weights needed for training conduct. Due to the overwhelm size of the orignial dataset and time/resource limitations, we are using a subset of the dataset to perform the logistic model training.

Double-click (or enter) to edit

```
# no code is required for this section
'''
if you want to use an image outside this notebook for explanaition,
you can upload it to your google drive and show it with OpenCV or matplotlib
'''
# mount this notebook to your google drive
drive.mount('/content/gdrive')

# define dirs to workspace and data
img_dir = '/content/gdrive/My Drive/Colab Notebooks/<path-to-your-image>'

import cv2
img = cv2.imread(img_dir)
cv2.imshow("Title", img)
```

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.

```
# import  packages you need
import numpy as np
from google.colab import drive
import pandas as pd
import tensorflow as tf
```

Data

The data we used in this research project is coming from the MIMIC III Health datasets which aligns with the research paper. The research paper used this dataset as a cross validation. However due to certain data being unavailble as the original dataset were from NYU CUIMC which we are unable to retrieve. Certain modifications are done to the dataset to make the usable in our context.

Double-click (or enter) to edit

```
# dir and function to load raw data
raw_data_dir = '/content/drive/MyDrive/Colab Notebooks/Data'

# The data is from MIMIC-III databases, I select 65 patients, tables I used are
# CHARTEVENTS and PATIENTS, I joined the two tables using other softwares.

def load_raw_data(raw_data_dir):
  # implement this function to load raw data to dataframe/numpy array/tensor
    df = pd.read_csv(raw_data_dir+'/master.csv')
    return df

raw_data = load_raw_data(raw_data_dir)
print(raw_data.head())

# calculate statistics
def calculate_stats(raw_data):
  # implement this function to calculate the statistics
  # it is encouraged to print out the results
  gender_counts = raw_data.groupby('gender')['subject_id'].nunique()
  itemid_stats = raw_data.groupby('subject_id').size()
  item_stat = raw_data.groupby('itemid').size()
  avg = itemid_stats.mean()
  max = itemid_stats.max()
  min = itemid_stats.min()
  return gender_counts,avg,max,min,item_stat
print(calculate_stats(raw_data))


# Data is preprocessed through other softwares into  three files
# demo_records.pkl: The first two value denotes male or female.
# [1,0] for male and [0,1] for female or vice versa.
# Age and Elixhauser index were normalized. Ex: [1, 0, 0.5, 0.4]
# patient_records.pkl: Drug exposure and diagnoses were one-hot encoded.
# Ex: [1, 0, 0, ..., 0, 1]
# labels.pkl: 1 indicates delirium


# 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)



       subject_id  itemid        dob       dod  age gender
    0           2     211  4/11/2025  1/0/1900   95      M
    1           2     834  4/11/2025  1/0/1900   95      M
    2           2     926  4/11/2025  1/0/1900   95      M
    3           2    3348  4/11/2025  1/0/1900   95      M
    4           2    3353  4/11/2025  1/0/1900   95      M
    (gender
    F    23
    M    42
    Name: subject_id, dtype: int64, 2355.9692307692308, 27945, 16, itemid
    1           1
    2           1
    3           1
    25          1
    26          3
              ...
    227466    126
    227467    104
    227468     20
    227471      6
    227516      1
    Length: 741, dtype: int64)


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

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.

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


#This is the implementation of the logistic regression model using TensorFlow.
#Consist of training the model with cross-validation, handling data loading, preprocessing, and saving model metrics.

import tensorflow as tf
import numpy as np
import pickle
import random
import os
from sklearn.metrics import roc_curve

#Extends Tensorflows
#Includes layers of concatenation of features and a dense layer of logistic regression wiht L2 regularization
class LogisticRegression(tf.keras.Model):
    def __init__(self, config):
        super(LogisticRegression, self).__init__()
        self.optimizer = tf.keras.optimizers.Adam(config["learning_rate"])

        self.concatenation = tf.keras.layers.Concatenate(axis=1, name="concatenation")
        self.lr = tf.keras.layers.Dense(1, activation=tf.keras.activations.sigmoid, name="lr",
        kernel_regularizer=tf.keras.regularizers.L2(l2=config["l2_reg"]))

    def call(self, x, d):
        x = unit_normalization(x)
        return self.lr(self.concatenation([x, d]))

#computes a negative log likelihood for a binary classifcation.
def compute_loss(model, x, d, label):
    prediction = model(x, d)
    loss_sum = tf.negative(tf.add(tf.multiply(5, tf.multiply(label, tf.math.log(prediction))),
                                  tf.multiply(tf.subtract(1., label), tf.math.log(tf.subtract(1., prediction)))))
    return tf.reduce_mean(loss_sum)

def calculate_auc(model, test_x, test_d, test_y, config):
    AUC = tf.keras.metrics.AUC(num_thresholds=200)
    AUC.reset_states()
    x, d, y = pad_matrix(test_x, test_d, test_y, config)
    pred = model(x, d)
    #print(pred)
    AUC.update_state(y, pred)

    return AUC.result().numpy()

def calculate_ROC(model, test_x, test_d, test_y, config):
    x, d, y = pad_matrix(test_x, test_d, test_y, config)
    pred = model(x,d)
    fpr, tpr, thresholds = roc_curve(test_y, pred)
    return fpr, tpr, thresholds

def calculate_ppv(model, test_x, test_d, test_y, config):
    ppv = tf.keras.metrics.Precision(thresholds=0.8)
    ppv.reset_states()
    x, d, y = pad_matrix(test_x, test_d, test_y, config)
    pred = model(x,d)
    ppv.update_state(y, pred)
    return ppv.result().numpy()

#to load, save shuffle and pad data
def load_data(patient_record_path, demo_record_path, labels_path):
    patient_record = pickle.load(open(patient_record_path, 'rb'))
    demo_record = pickle.load(open(demo_record_path, 'rb'))
    labels = pickle.load(open(labels_path, 'rb'))

    return patient_record, demo_record, labels

def save_data(output_path, mydata):
```

```
    def save_data(output_path, mydata):
        with open(output_path, 'wb') as f:
            pickle.dump(mydata, f)

    def pad_matrix(records, demos, labels, config):
        n_patients = len(records)
        #input_vocabsize = config["input_vocabsize"]
        #demo_vocabsize = config["demo_vocabsize"]

        x = np.array(records).astype(np.float32) # sum of all visits of the patient
        d = np.array(demos).astype(np.float32)
        y = np.array(labels).astype(np.float32)

        #for idx, rec in enumerate(records):
            #for visit in rec:
                #x[idx, visit] += 1

        #x = np.clip(0, 1, x) # clip values bigger than 1.

        #for idx, demo in enumerate(demos):
            #d[idx, int(demo[:-2])] = 1. # the last element of demos is age
            #d[idx, -1:] = demo[-1:]

        return x, d, y

    def shuffle_data(data1, data2, data3):
        data1, data2, data3 = np.array(data1), np.array(data2), np.array(data3)
        idx = np.arange(len(data1))
        random.seed(1234)
        random.shuffle(idx)

        return data1[idx], data2[idx], data3[idx]

    def unit_normalization(myarray):
        avg = tf.reshape(tf.math.reduce_mean(myarray, axis=-1), shape=(myarray.shape[0], 1))
        std = tf.reshape(tf.math.reduce_std(myarray, axis=-1), shape=(myarray.shape[0], 1))
        return tf.math.divide(tf.math.subtract(myarray, avg), std)
    def train_lreg_kfold(output_path, patient_record_path, demo_record_path, labels_path, epochs, batch_size,
                 input_vocabsize, demo_vocabsize, l2_reg=0.001, learning_rate=0.001, k=5, times =5, notes=None):

        tf.random.set_seed(1234)
        config = locals().copy()
        print(config["input_vocabsize"])
        for i in range(times):
            version = i
            print("load data...")
            recs, demos, labels = load_data(patient_record_path, demo_record_path, labels_path)

            print("split the dataset into k-fold...")
            recs, demos, labels = shuffle_data(recs, demos, labels)
            chunk_size = int(np.floor(len(labels) / k))
            np.split(np.arange(len(labels)), [chunk_size*i for i in range(k)])
            folds = np.tile(np.split(np.arange(len(labels)), [chunk_size*i for i in range(int(k))])[1:], 2)
            print(len(folds))


            k_fold_auc = []
            k_fold_ppv = []
            k_fold_tpr = []
            mean_fpr = np.linspace(0,1,200)
            k_fold_training_loss = []

            for i in range(k):
                train_x, test_x = recs[np.concatenate([folds[j] for j in range(k) if j != i % k])], recs[folds[(i%k)]]
                train_d, test_d = demos[np.concatenate([folds[j] for j in range(k) if j != i % k])], demos[folds[(i%k)]]
                train_y, test_y = labels[np.concatenate([folds[j] for j in range(k) if j != i % k])], labels[folds[(i%k)]]
                print(len(train_y))
                print(len(test_y))

                num_batches = int(np.ceil(float(len(train_x)) / float(batch_size)))
                training_loss = []

                print("build and initialize model for {k}th fold...".format(k=i+1))
                lr_model = LogisticRegression(config)
                #_ = lr_model(train_x, train_d)
                #print(lr_model)

                best_auc = 0
```

```
            best_epoch = 0
            best_model =  None
            #print(best_model)
            print("start training...")
            for epoch in range(epochs):
                loss_record = []
                progbar = tf.keras.utils.Progbar(num_batches)

                for t in random.sample(range(num_batches), num_batches):
                    batch_x = train_x[t * batch_size:(t+1) * batch_size]
                    batch_d = train_d[t * batch_size:(t+1) * batch_size]
                    batch_y = train_y[t * batch_size:(t+1) * batch_size]

                    x, d, y = pad_matrix(batch_x, batch_d, batch_y, config)

                    with tf.GradientTape() as tape:
                        batch_cost = compute_loss(lr_model, x, d, y)
                    gradients = tape.gradient(batch_cost, lr_model.trainable_variables)
                    lr_model.optimizer.apply_gradients(zip(gradients, lr_model.trainable_variables))

                    loss_record.append(batch_cost.numpy())
                    progbar.add(1)

                print('epoch:{e}, mean loss:{l:.6f}'.format(e=epoch+1, l=np.mean(loss_record)))
                training_loss.append(np.mean(loss_record))
                current_auc = calculate_auc(lr_model, test_x, test_d, test_y, config)
                print(current_auc)
                #print(lr_model.get_weights())
                #print('epoch:{e}, current auc:{l:.6f}'.format(e=epoch+1, l=current_auc))
                if current_auc > best_auc:
                    best_auc = current_auc
                    best_epoch = epoch+1
                    best_model = lr_model.get_weights()

            k_fold_training_loss.append(training_loss)
            print("calculate AUC on the best model using the test set")
            lr_model.set_weights(best_model)
            test_auc = calculate_auc(lr_model, test_x, test_d, test_y, config)
            test_ppv = calculate_ppv(lr_model, test_x, test_d, test_y, config)
            print("AUC of {k}th fold: {auc:.6f}".format(k=i+1, auc=test_auc))
            print("PPV of {k}th fold: {ppv:.6f}".format(k=i+1, ppv=test_ppv))
            k_fold_auc.append(test_auc)
            k_fold_ppv.append(test_ppv)
            fpr, tpr, thresholds = calculate_ROC(lr_model, test_x, test_d, test_y, config)
            k_fold_tpr.append(np.interp(mean_fpr, fpr, tpr))

        print("saving k-fold results...")
        mode_name = "mhot"
        #np.save(os.path.join(output_path, "LRS_{m}_{k}fold_l{l}_training_loss_ver{i}.npy".format(k=k, m=mode_name, l=learning_ra
        np.save(os.path.join(output_path, "LRS_{m}_{k}fold_l{l}_auc_ver{i}.npy".format(k=k, m=mode_name, l=learning_rate, i=versi
        np.save(os.path.join(output_path, "LRS_{m}_{k}fold_l{l}_tpr_ver{i}.npy".format(k=k, m=mode_name, l=learning_rate, i=versi
        np.save(os.path.join(output_path, "LRS_{m}_{k}fold_l{l}_ppv_ver{i}.npy".format(k=k, m=mode_name, l=learning_rate, i=versi
        #np.save(os.path.join(output_path, "LRS_{m}_e{e}_l{l}_model_ver{i}.npy".format(m=mode_name, e=epochs, l=learning_rate, i=

    save_data(os.path.join(output_path, "LRS_{m}_{k}fold_l{l}_config.pkl".format(k=k, m=mode_name, l=learning_rate)), config)

train_lreg_kfold('/content/drive/MyDrive/Colab Notebooks/Data/ml_output', '/content/drive/MyDrive/Colab Notebooks/Data/patient_re
```

```
100
load data...
split the dataset into k-fold...
5
104
26
build and initialize model for 1th fold...
start training...
52/52 [==============================] - 1s 14ms/step
epoch:1, mean loss:2.390759
0.55
52/52 [==============================] - 1s 15ms/step
epoch:2, mean loss:2.082414
0.55
52/52 [==============================] - 1s 15ms/step
epoch:3, mean loss:1.882113
0.5625
52/52 [==============================] - 1s 16ms/step
epoch:4, mean loss:1.750731
0.575
```

```
52/52 [==============================] - 1s 17ms/step
epoch:5, mean loss:1.647370
0.625
52/52 [==============================] - 1s 11ms/step
epoch:6, mean loss:1.574192
0.6375
52/52 [==============================] - 1s 11ms/step
epoch:7, mean loss:1.502979
0.65000004
52/52 [==============================] - 1s 11ms/step
epoch:8, mean loss:1.449516
0.65
52/52 [==============================] - 1s 11ms/step
epoch:9, mean loss:1.402550
0.65
52/52 [==============================] - 1s 11ms/step
epoch:10, mean loss:1.359486
0.65
52/52 [==============================] - 1s 11ms/step
epoch:11, mean loss:1.321329
0.65
52/52 [==============================] - 1s 11ms/step
epoch:12, mean loss:1.292237
0.65
52/52 [==============================] - 1s 11ms/step
epoch:13, mean loss:1.257666
0.65
52/52 [==============================] - 1s 11ms/step
epoch:14, mean loss:1.233697
0.625
52/52 [==============================] - 1s 11ms/step
epoch:15, mean loss:1.209484
0.625
52/52 [==============================] - 1s 11ms/step
epoch:16, mean loss:1.190927
0.63750005
52/52 [==============================] - 1s 11ms/step
epoch:17, mean loss:1.169707
```

Results

In this draft version of the project implementation we noticed from the graph below we are getting a score of 0.708 AUC instead of the 0.87 AUC achieved in the research paper. We analyzed this difference is due to the dataset limitation as we do not have access to the NYU UCIMC, we will increase the sub-dataset from the MIMIC III in the next two weeks and continuing to tweak the parameters to achieve higher AUC.

But overall according to this 0.708 AUC it sugguests that the training model discussed in the paper has good discriminative ability. It means that there is a 78% chance that the model will be able to distinguish between the positive class and negative class correctly.

Therefore in this draft version of the project, our goal is to continue tweak the parameters and increase sub-dataset size to achieve higher AUC as they did in the research paper.

So far this model has a reasonbly strong ability to classify the positive and negative cases correctly across different thresholds. It is a valuable metric when deciding whether the model's predictive accuracy is sufficient for pratical applications.

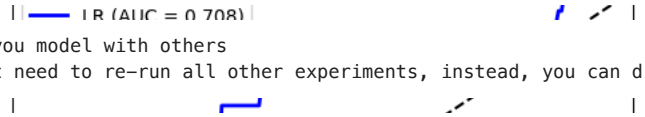
```
# metrics to evaluate my model

# plot figures to better show the results

# it is better to save the numbers and figures for your presentation.
import glob
import matplotlib.pyplot as plt
tpr_LRS = np.load('/content/drive/MyDrive/Colab Notebooks/Data/ml_output/LRS_mhot_5fold_l0.001_tpr_ver4.npy')
auc_LRS = np.load('/content/drive/MyDrive/Colab Notebooks/Data/ml_output/LRS_mhot_5fold_l0.001_auc_ver4.npy')
mean_tpr_LRS = np.mean(tpr_LRS, axis=0)
mean_auc_LRS = np.mean(auc_LRS, axis=0)
mean_fpr = np.linspace(0,1,200)
f = plt.figure(figsize=(6, 4))

plt.figure(1)
plt.xlim(0,1)
plt.ylim(0,1)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(mean_fpr, mean_tpr_LRS, color='blue',label=r'LR (AUC = %0.3f)' % (mean_auc_LRS),lw=2, alpha=1)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.legend(loc="best")
plt.show()
f.savefig("./ROC all.pdf", bbox_inches='tight')
```

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
```

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
'''
import numpy as np
```

References

1. Sun, J, [paper title], [journal title], [year], [volume]:[issue], doi: [doi link to paper]

Feel free to add new sections