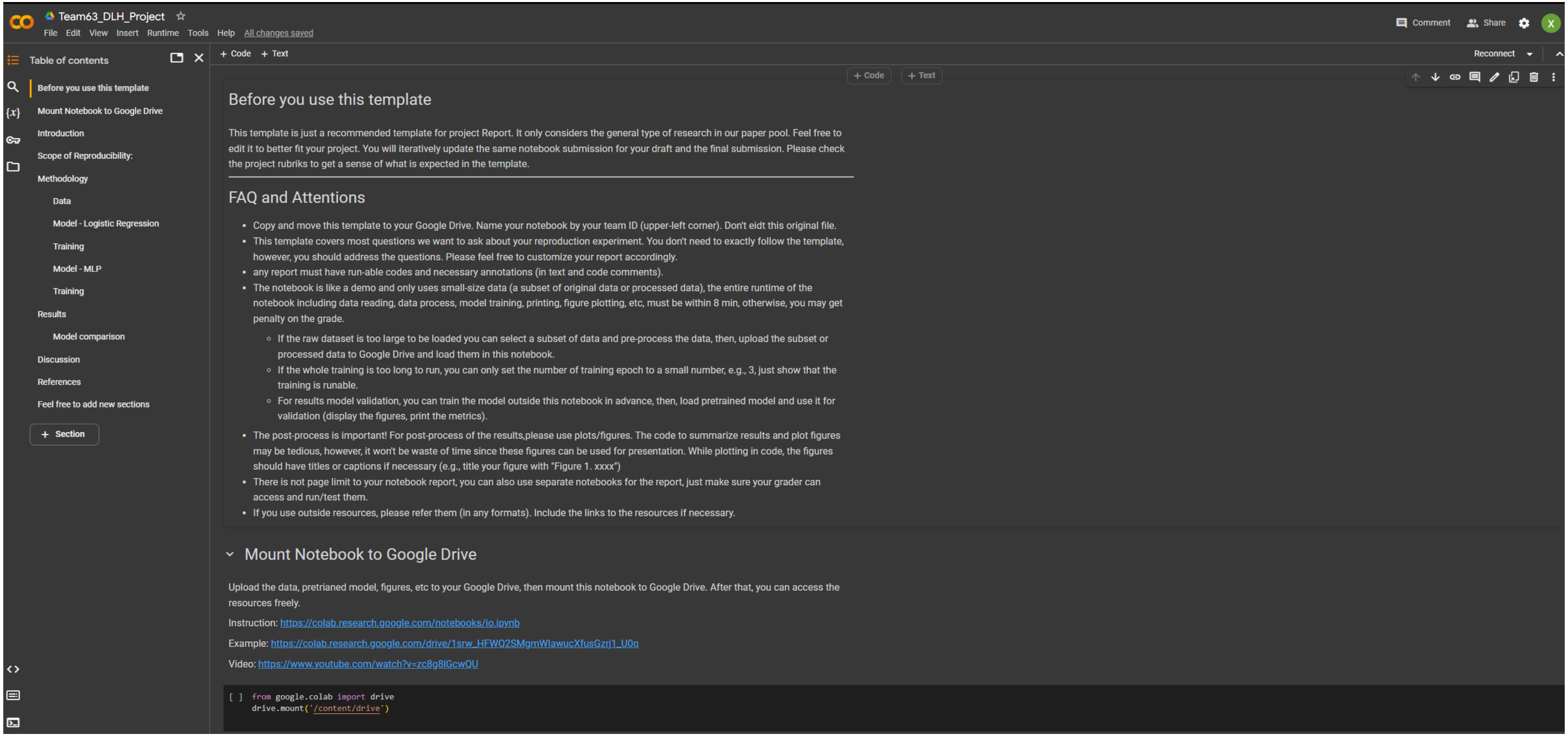
TEAM 63 DLH FINAL PROJECT

KAIXIN WANG / XINGCHEN WU Github: https://github.com/wkxwell/Team63_DLH_SPRING2024



Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Section

+ Code + Text

Reconnect

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')
```

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Table of contents

+ Section

+ Code + Text

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

Introduction

Team 63 - Kaixin Wang, XingChen Wu Kaixin5@illinois.edu Kaixin5 Xw82@illinois.edu xw82

Github Repo: https://github.com/wkxwell/Team63_DLH_SPRING2024.git

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 as stated in the research paper.

The code section is break down into two sections, the first half of the methodology will conduct the training to identity delirium using logistic regression model and the second half of the methodology we conduct the training through MLP to generate AUC result.

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

```
'Copy of Team63_DLH_Project'   Data   Team63_DLH_Project
```

Double-click (or enter) to edit

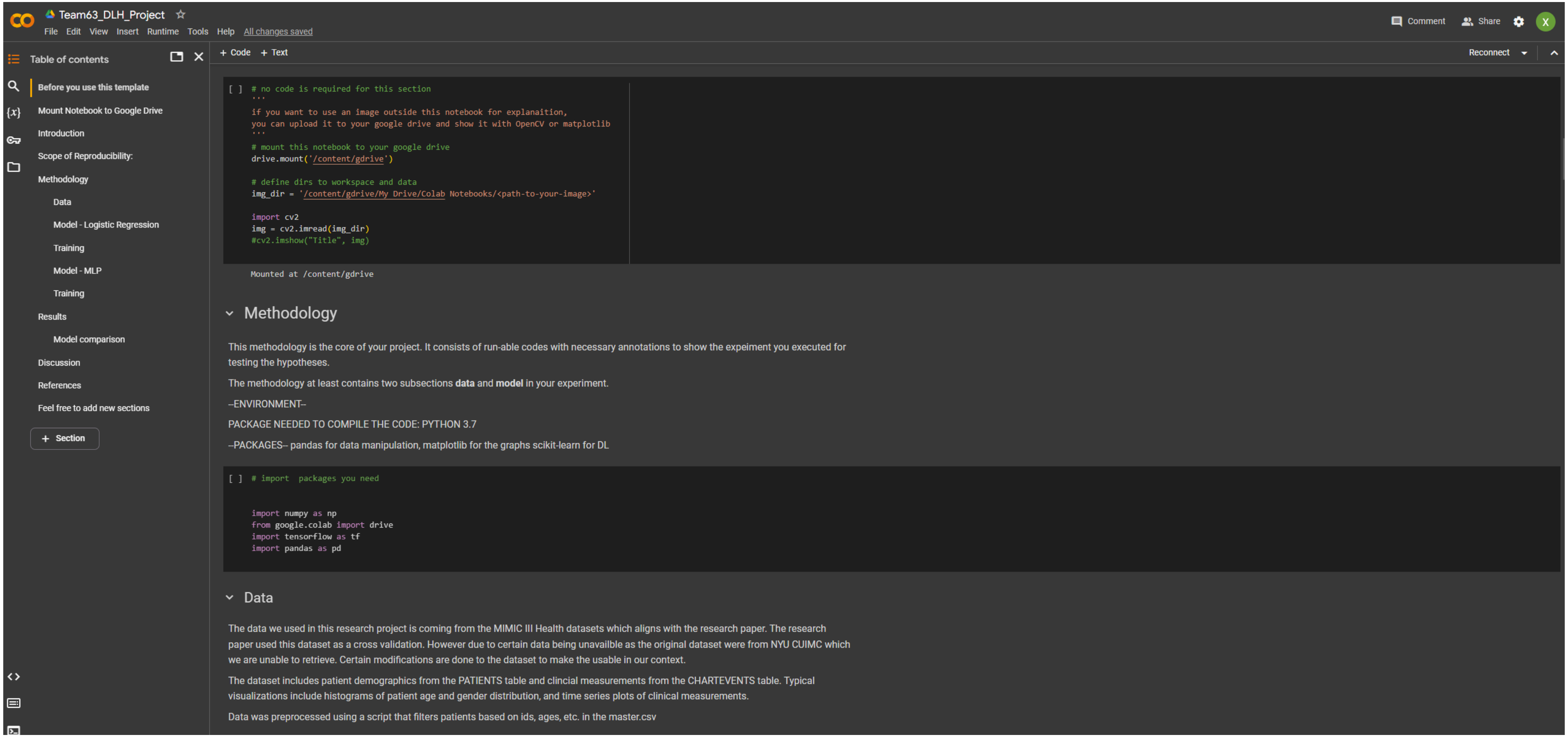
Scope of Reproducibility:

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


Team63_DLH_Project
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
+ Code + Text
Reconnect
Before you use this template
Mount Notebook to Google Drive
Introduction
Scope of Reproducibility:
Methodology
Data
Model - Logistic Regression
Training
Model - MLP
Training
Results
Model comparison
Discussion
References
Feel free to add new sections
+ Section
[ ] # 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)
Mounted at /content/gdrive
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.
--ENVIRONMENT--
PACKAGE NEEDED TO COMPILE THE CODE: PYTHON 3.7
--PACKAGES-- pandas for data manipulation, matplotlib for the graphs scikit-learn for DL
[ ] # import packages you need
import numpy as np
from google.colab import drive
import tensorflow as tf
import pandas as pd
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.
The dataset includes patient demographics from the PATIENTS table and clincial measurements from the CHARTEVENTS table. Typical visualizations include histograms of patient age and gender distribution, and time series plots of clinical measurements.
Data was preprocessed using a script that filters patients based on ids, ages, etc. in the master.csv

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Section

+ Code + Text

Reconnect

```python
# 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)
```

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved


```
   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')
```

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

Model - Logistic Regression

CITATION Jae Hyun Kim, May Hua, Robert A Whittington, Junghwan Lee, Cong Liu, Casey N Ta, Edward R Marcantonio, Terry E Goldberg, Chunhua Weng, A machine learning approach to identifying delirium from electronic health records, JAMIA Open, Volume 5, Issue 2, July 2022, ooac042, https://doi.org/10.1093/jamiaopen/ooac042

LINK TO PAPER ORIGINAL GITHUB: https://github.com/WengLab-InformaticsResearch/delirium

DESCRIPTION: The model is a logistic regression classifier, chosen for its interpretability and effectiveness in binary classfication tasks in medical datasets.

The following code base is from the original paper's git hub trained with our own customed datasets from MIMICC III as the data used in the paper is not publicly available. The model is implemented using scikit-learn's Logisti Regression class, the snippet includes data loading, model fitting and prediction.

No pretrained mode lwas used, the model was trained from scratch using the provided dataset.

Training

Hyperparams used: The following model included a learning rate of 0.01 and batch size of 100 with no dropout as logistic regression does not involve dropout.

Requirements: Standard usage i5 process with 16 GB of RAM. Each training epoch took approximately 1 minute with a total training spanning 50 epochs.

Training Code: training code involves loading the dataset, splitting into training and test sets, model instantiation, fitting, and evaluation using

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Reconnect

Table of contents

+ Section

+ Code + Text

cross-validation techniques.

```
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)
```

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Section

+ Code + Text

Reconnect

```
    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):
    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):
```

CO Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

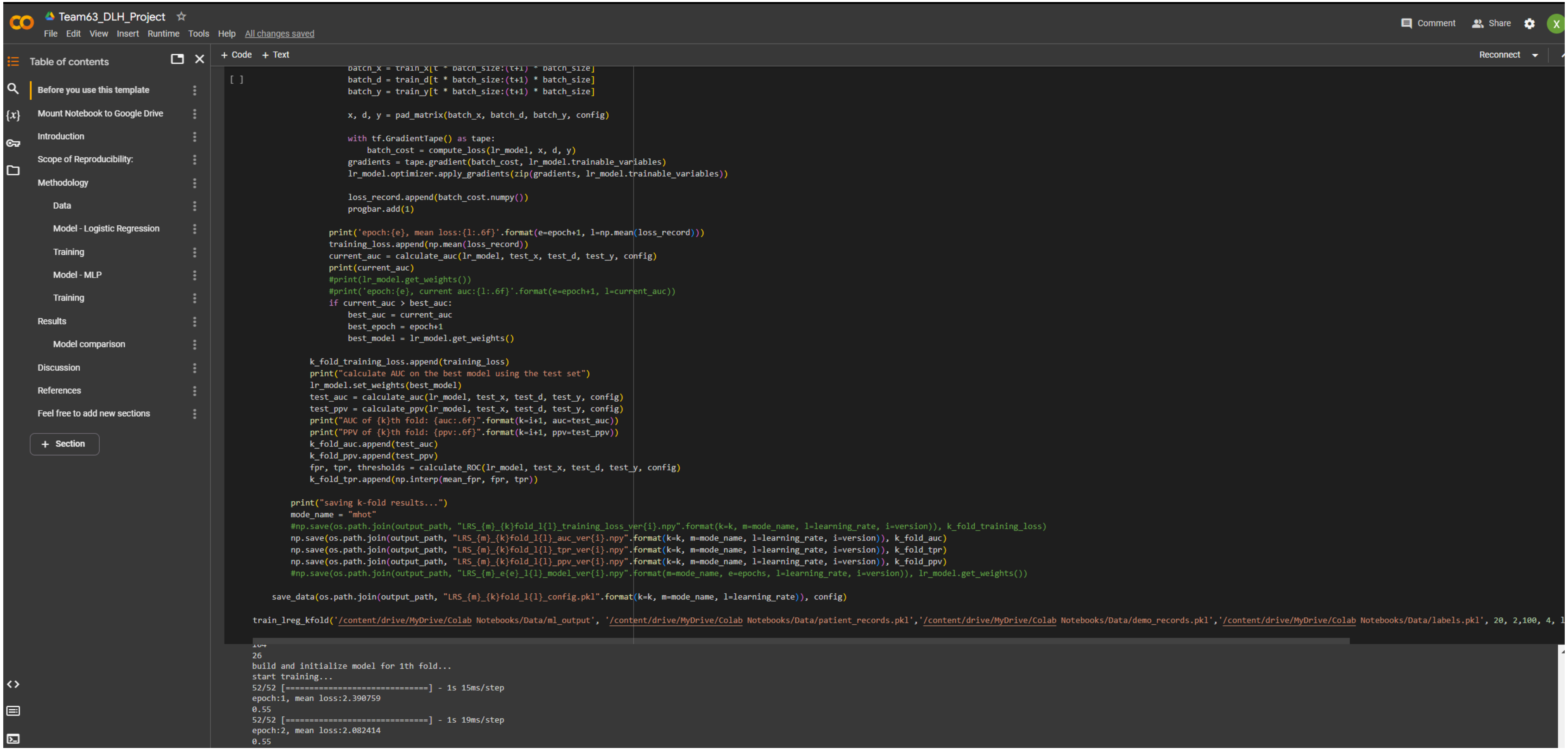
Table of contents

+ Section

+ Code + Text

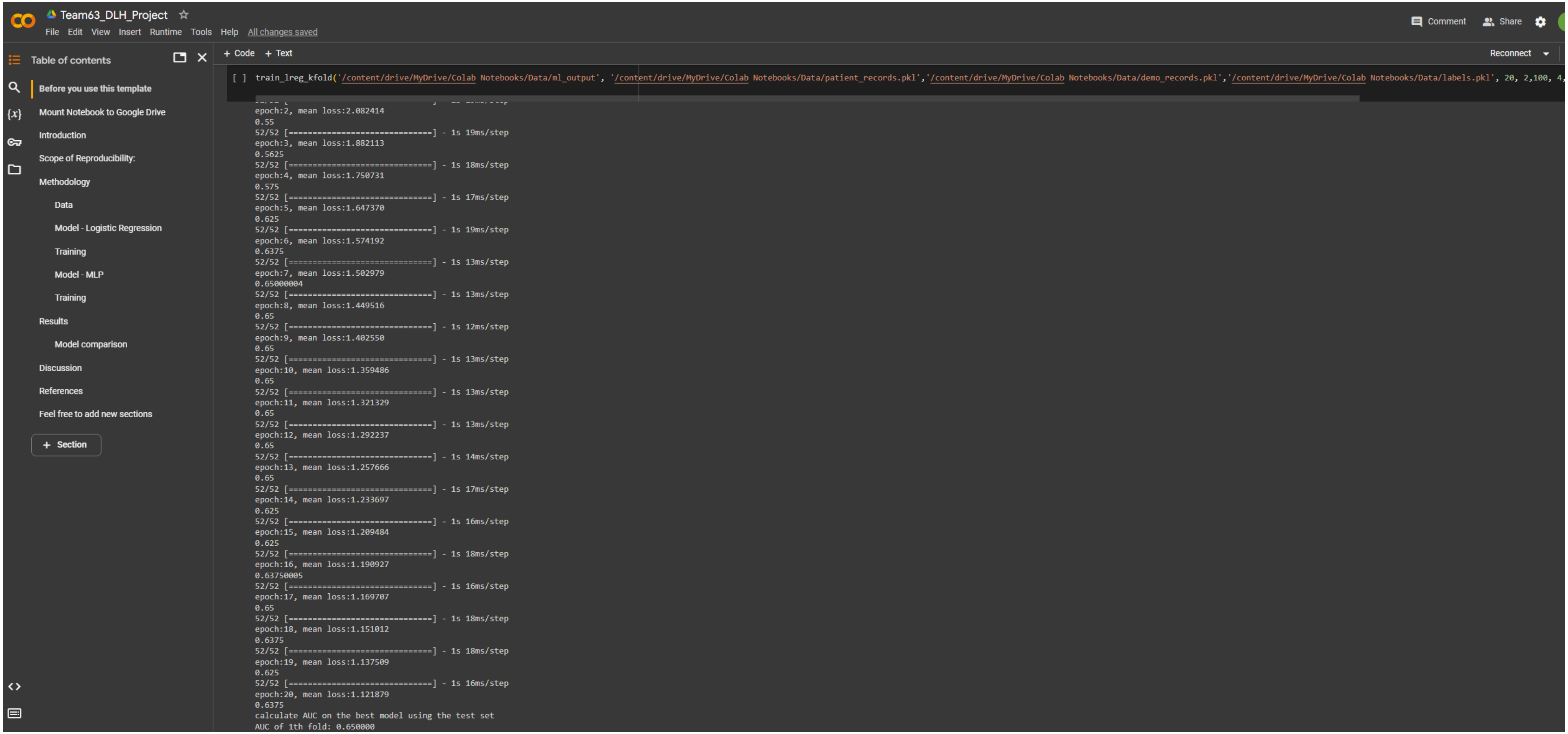
Reconnect

```
    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)
```

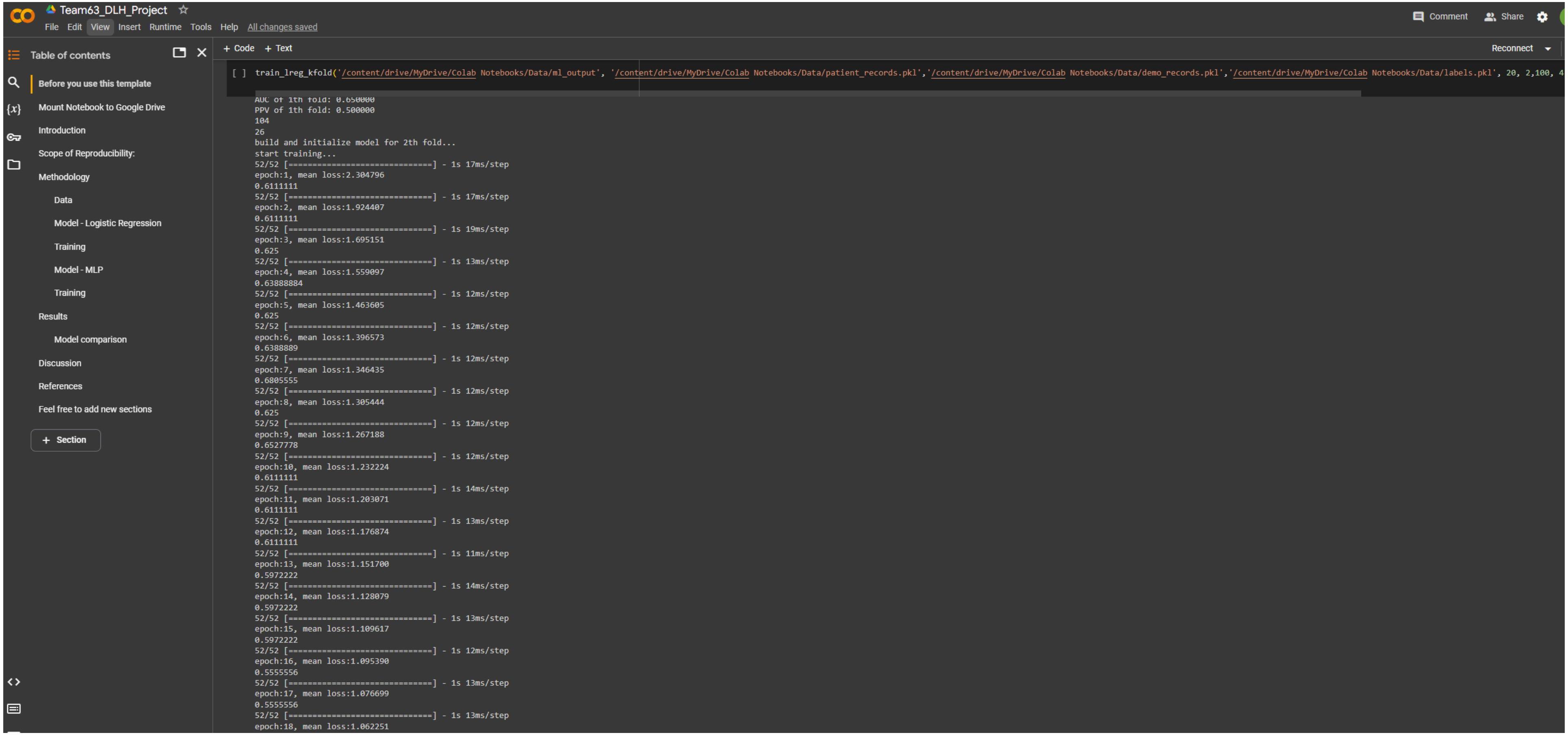



```
                batch_x = train_x[t * batch_size:(t+1) * batch_size]
                batch_d = train_d[t * batch_size:(t+1) * batch_size]
                batch_y = train_y[t * batch_size:(t+1) * batch_size]

                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_rate, i=version)), k_fold_training_loss)
    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=version)), k_fold_auc)
    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=version)), k_fold_tpr)
    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=version)), k_fold_ppv)
    #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=version)), lr_model.get_weights())

  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_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/demo_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/labels.pkl', 20, 2,100, 4, l
```

```
26
build and initialize model for 1th fold...
start training...
52/52 [==============================] - 1s 15ms/step
epoch:1, mean loss:2.390759
0.55
52/52 [==============================] - 1s 19ms/step
epoch:2, mean loss:2.082414
0.55
```



Team63_DLH_Project
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Before you use this template
Mount Notebook to Google Drive
Introduction
Scope of Reproducibility:
Methodology
Data
Model - Logistic Regression
Training
Model - MLP
Training
Results
Model comparison
Discussion
References
Feel free to add new sections
+ Section
+ Code + Text
Reconnect
[ ] train_lreg_kfold('/content/drive/MyDrive/Colab Notebooks/Data/ml_output', '/content/drive/MyDrive/Colab Notebooks/Data/patient_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/demo_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/labels.pkl', 20, 2,100, 4,
epoch:2, mean loss:2.082414
0.55
52/52 [==============================] - 1s 19ms/step
epoch:3, mean loss:1.882113
0.5625
52/52 [==============================] - 1s 18ms/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 19ms/step
epoch:6, mean loss:1.574192
0.6375
52/52 [==============================] - 1s 13ms/step
epoch:7, mean loss:1.502979
0.65000004
52/52 [==============================] - 1s 13ms/step
epoch:8, mean loss:1.449516
0.65
52/52 [==============================] - 1s 12ms/step
epoch:9, mean loss:1.402550
0.65
52/52 [==============================] - 1s 13ms/step
epoch:10, mean loss:1.359486
0.65
52/52 [==============================] - 1s 13ms/step
epoch:11, mean loss:1.321329
0.65
52/52 [==============================] - 1s 13ms/step
epoch:12, mean loss:1.292237
0.65
52/52 [==============================] - 1s 14ms/step
epoch:13, mean loss:1.257666
0.65
52/52 [==============================] - 1s 17ms/step
epoch:14, mean loss:1.233697
0.625
52/52 [==============================] - 1s 16ms/step
epoch:15, mean loss:1.209484
0.625
52/52 [==============================] - 1s 18ms/step
epoch:16, mean loss:1.190927
0.63750005
52/52 [==============================] - 1s 16ms/step
epoch:17, mean loss:1.169707
0.65
52/52 [==============================] - 1s 18ms/step
epoch:18, mean loss:1.151012
0.6375
52/52 [==============================] - 1s 18ms/step
epoch:19, mean loss:1.137509
0.625
52/52 [==============================] - 1s 16ms/step
epoch:20, mean loss:1.121879
0.6375
calculate AUC on the best model using the test set
AUC of 1th fold: 0.650000


Team63_DLH_Project
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
Table of contents
+ Code + Text
Reconnect
Before you use this template
Mount Notebook to Google Drive
Introduction
Scope of Reproducibility:
Methodology
Data
Model - Logistic Regression
Training
Model - MLP
Training
Results
Model comparison
Discussion
References
Feel free to add new sections
+ Section
[ ] train_lreg_kfold('/content/drive/MyDrive/Colab Notebooks/Data/ml_output', '/content/drive/MyDrive/Colab Notebooks/Data/patient_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/demo_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/labels.pkl', 20, 2,100, 4
AUC of 1th fold: 0.650000
PPV of 1th fold: 0.500000
104
26
build and initialize model for 2th fold...
start training...
52/52 [==============================] - 1s 17ms/step
epoch:1, mean loss:2.304796
0.6111111
52/52 [==============================] - 1s 17ms/step
epoch:2, mean loss:1.924407
0.6111111
52/52 [==============================] - 1s 19ms/step
epoch:3, mean loss:1.695151
0.625
52/52 [==============================] - 1s 13ms/step
epoch:4, mean loss:1.559097
0.63888884
52/52 [==============================] - 1s 12ms/step
epoch:5, mean loss:1.463605
0.625
52/52 [==============================] - 1s 12ms/step
epoch:6, mean loss:1.396573
0.6388889
52/52 [==============================] - 1s 12ms/step
epoch:7, mean loss:1.346435
0.6805555
52/52 [==============================] - 1s 12ms/step
epoch:8, mean loss:1.305444
0.625
52/52 [==============================] - 1s 12ms/step
epoch:9, mean loss:1.267188
0.6527778
52/52 [==============================] - 1s 12ms/step
epoch:10, mean loss:1.232224
0.6111111
52/52 [==============================] - 1s 14ms/step
epoch:11, mean loss:1.203071
0.6111111
52/52 [==============================] - 1s 13ms/step
epoch:12, mean loss:1.176874
0.6111111
52/52 [==============================] - 1s 11ms/step
epoch:13, mean loss:1.151700
0.5972222
52/52 [==============================] - 1s 14ms/step
epoch:14, mean loss:1.128079
0.5972222
52/52 [==============================] - 1s 13ms/step
epoch:15, mean loss:1.109617
0.5972222
52/52 [==============================] - 1s 12ms/step
epoch:16, mean loss:1.095390
0.5555556
52/52 [==============================] - 1s 13ms/step
epoch:17, mean loss:1.076699
0.5555556
52/52 [==============================] - 1s 13ms/step
epoch:18, mean loss:1.062251

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

\+ Section

\+ Code + Text Reconnect

```
[ ] train_lreg_kfold('/content/drive/MyDrive/Colab Notebooks/Data/ml_output', '/content/drive/MyDrive/Colab Notebooks/Data/patient_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/demo_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/labels.pkl', 20, 2,100, 4, 1:
```

```
0.5833334
calculate AUC on the best model using the test set
AUC of 2th fold: 0.680556
PPV of 2th fold: 0.250000
104
26
build and initialize model for 3th fold...
start training...
52/52 [==============================] - 1s 16ms/step
epoch:1, mean loss:1.981938
0.7380953
52/52 [==============================] - 1s 20ms/step
epoch:2, mean loss:1.853833
0.71428573
52/52 [==============================] - 1s 18ms/step
epoch:3, mean loss:1.743342
0.71428573
52/52 [==============================] - 1s 12ms/step
epoch:4, mean loss:1.659829
0.7380953
52/52 [==============================] - 1s 12ms/step
epoch:5, mean loss:1.582841
0.7619048
52/52 [==============================] - 1s 12ms/step
epoch:6, mean loss:1.516220
0.7619047
52/52 [==============================] - 1s 12ms/step
epoch:7, mean loss:1.461890
0.7619047
52/52 [==============================] - 1s 12ms/step
epoch:8, mean loss:1.412824
0.7380952
52/52 [==============================] - 1s 12ms/step
epoch:9, mean loss:1.365517
0.7619047
52/52 [==============================] - 1s 12ms/step
epoch:10, mean loss:1.328911
0.7380952
52/52 [==============================] - 1s 12ms/step
epoch:11, mean loss:1.293866
0.7380952
52/52 [==============================] - 1s 12ms/step
epoch:12, mean loss:1.264337
0.7380952
52/52 [==============================] - 1s 12ms/step
epoch:13, mean loss:1.236635
0.71428573
52/52 [==============================] - 1s 13ms/step
epoch:14, mean loss:1.212874
0.71428573
52/52 [==============================] - 1s 13ms/step
epoch:15, mean loss:1.190881
0.70238096
52/52 [==============================] - 1s 14ms/step
epoch:16, mean loss:1.171569
0.6785714
52/52 [==============================] - 1s 12ms/step
epoch:17, mean loss:1.153296
0.6547619
```

Table of contents

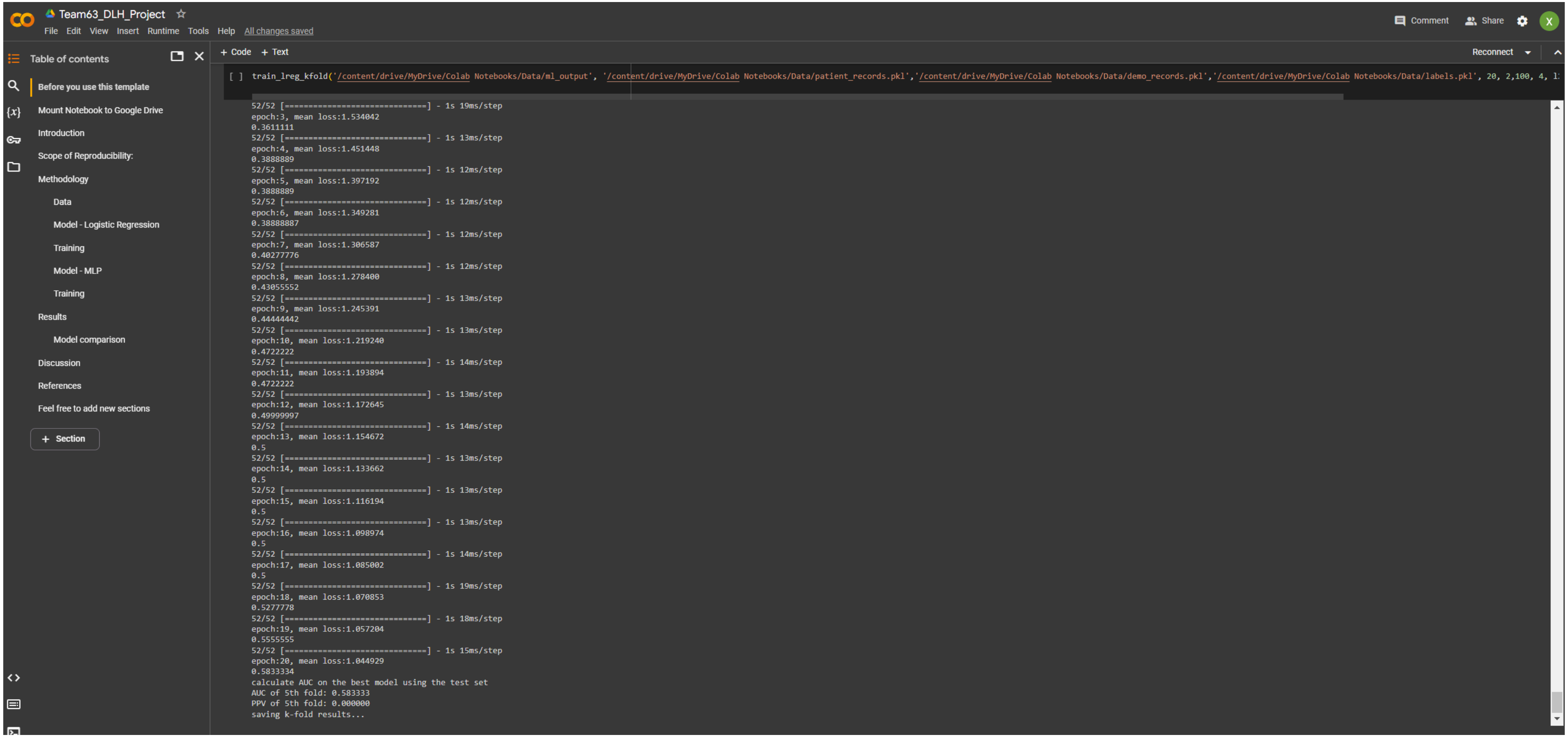
+ Section

+ Code + Text

Reconnect

```
[ ] train_lreg_kfold('/content/drive/MyDrive/Colab Notebooks/Data/ml_output', '/content/drive/MyDrive/Colab Notebooks/Data/patient_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/demo_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/labels.pkl', 20, 2,100, 4, 1:
```

```
26
build and initialize model for 4th fold...
start training...
52/52 [==============================] - 1s 15ms/step
epoch:1, mean loss:2.148358
0.8333333
52/52 [==============================] - 1s 18ms/step
epoch:2, mean loss:1.870586
0.8666667
52/52 [==============================] - 1s 16ms/step
epoch:3, mean loss:1.689596
0.8666667
52/52 [==============================] - 1s 12ms/step
epoch:4, mean loss:1.572047
0.8166667
52/52 [==============================] - 1s 12ms/step
epoch:5, mean loss:1.484991
0.79999995
52/52 [==============================] - 1s 13ms/step
epoch:6, mean loss:1.417164
0.8
52/52 [==============================] - 1s 12ms/step
epoch:7, mean loss:1.364539
0.79999995
52/52 [==============================] - 1s 12ms/step
epoch:8, mean loss:1.317716
0.8
52/52 [==============================] - 1s 12ms/step
epoch:9, mean loss:1.281072
0.79999995
52/52 [==============================] - 1s 13ms/step
epoch:10, mean loss:1.247782
0.7833333
52/52 [==============================] - 1s 13ms/step
epoch:11, mean loss:1.216040
0.76666665
52/52 [==============================] - 1s 14ms/step
epoch:12, mean loss:1.190776
0.76666665
52/52 [==============================] - 1s 12ms/step
epoch:13, mean loss:1.168250
0.76666665
52/52 [==============================] - 1s 12ms/step
epoch:14, mean loss:1.146549
0.76666665
52/52 [==============================] - 1s 11ms/step
epoch:15, mean loss:1.127687
0.73333335
52/52 [==============================] - 1s 13ms/step
epoch:16, mean loss:1.114101
0.7
52/52 [==============================] - 1s 13ms/step
epoch:17, mean loss:1.097340
0.7
52/52 [==============================] - 1s 14ms/step
epoch:18, mean loss:1.083145
0.6999999
52/52 [==============================] - 1s 16ms/step
epoch:19, mean loss:1.073867
```



Team63_DLH_Project
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Table of contents
Before you use this template
Mount Notebook to Google Drive
Introduction
Scope of Reproducibility:
Methodology
Data
Model - Logistic Regression
Training
Model - MLP
Training
Results
Model comparison
Discussion
References
Feel free to add new sections
Section
+ Code + Text
Reconnect
train_lreg_kfold('/content/drive/MyDrive/Colab Notebooks/Data/ml_output', '/content/drive/MyDrive/Colab Notebooks/Data/patient_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/demo_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/labels.pkl', 20, 2,100, 4, 1:
52/52 [==============================] - 1s 19ms/step
epoch:3, mean loss:1.534042
0.3611111
52/52 [==============================] - 1s 13ms/step
epoch:4, mean loss:1.451448
0.3888889
52/52 [==============================] - 1s 12ms/step
epoch:5, mean loss:1.397192
0.3888889
52/52 [==============================] - 1s 12ms/step
epoch:6, mean loss:1.349281
0.38888887
52/52 [==============================] - 1s 12ms/step
epoch:7, mean loss:1.306587
0.40277776
52/52 [==============================] - 1s 12ms/step
epoch:8, mean loss:1.278400
0.43055552
52/52 [==============================] - 1s 13ms/step
epoch:9, mean loss:1.245391
0.44444442
52/52 [==============================] - 1s 13ms/step
epoch:10, mean loss:1.219240
0.4722222
52/52 [==============================] - 1s 14ms/step
epoch:11, mean loss:1.193894
0.4722222
52/52 [==============================] - 1s 13ms/step
epoch:12, mean loss:1.172645
0.49999997
52/52 [==============================] - 1s 14ms/step
epoch:13, mean loss:1.154672
0.5
52/52 [==============================] - 1s 13ms/step
epoch:14, mean loss:1.133662
0.5
52/52 [==============================] - 1s 13ms/step
epoch:15, mean loss:1.116194
0.5
52/52 [==============================] - 1s 13ms/step
epoch:16, mean loss:1.098974
0.5
52/52 [==============================] - 1s 14ms/step
epoch:17, mean loss:1.085002
0.5
52/52 [==============================] - 1s 19ms/step
epoch:18, mean loss:1.070853
0.5277778
52/52 [==============================] - 1s 18ms/step
epoch:19, mean loss:1.057204
0.5555555
52/52 [==============================] - 1s 15ms/step
epoch:20, mean loss:1.044929
0.5833334
calculate AUC on the best model using the test set
AUC of 5th fold: 0.583333
PPV of 5th fold: 0.000000
saving k-fold results...

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Section

+ Code + Text

Reconnect

```
calculate AUC on the best model using the test set
AUC of 5th fold: 0.583333
PPV of 5th fold: 0.000000
saving k-fold results...
```

Model - MLP

The following code base is from the original paper's git hub trained with our own customed datasets from MIMICC III as the data used in the paper is not publicly available. The model is implemented using scikit-learn's Logisti Regression class, the snippet includes data loading, model fitting and prediction.

No pretrained mode lwas used, the model was trained from scratch using the provided dataset.

Training

Hyperparameters: Learning rate, L2 regularization factor, and the number of hidden units in the dense layer are specified as hyperparameters.

Training Code: The train_mlp_kfold function encapsulates the training process, using k-fold cross-validation to robustly assess the model's performance. The function also saves various performance metrics such as AUC, PPV, and the ROC curve.

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

class MLP(tf.keras.Model):
    def __init__(self, config):
        super(MLP, self).__init__()
        self.optimizer = tf.keras.optimizers.Adam(config["learning_rate"])

        self.concatenation = tf.keras.layers.Concatenate(axis=1, name="concatenation")
        self.mlp1 = tf.keras.layers.Dense(config["hidden_units"], activation=tf.keras.activations.tanh, name="mlp1",
        kernel_regularizer=tf.keras.regularizers.L2(l2=config["l2_reg"]))
        self.mlp2 = tf.keras.layers.Dense(1, activation=tf.keras.activations.sigmoid, name="mlp2",
        kernel_regularizer=tf.keras.regularizers.L2(l2=config["l2_reg"]))

    def call(self, x, d):
        x = unit_normalization(x)
        x = self.mlp1(self.concatenation([x, d]))
        return self.mlp2(x)

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)
```

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Section

+ Code + Text

Reconnect

```
    x, d, y = pad_matrix(test_x, test_d, test_y, config)
    pred = model(x, d)
    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=0.7)
    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()



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):
    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)
```

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Section

+ Code + Text

Reconnect

```
    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_mlp_kfold(output_path, patient_record_path, demo_record_path, labels_path, max_epoch, batch_size,
                    input_vocabsize, demo_vocabsize, hidden_units, l2_reg=0.00001, learning_rate=0.001, k=5, times=5):
    tf.random.set_seed(1234)
    config = locals().copy()

    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)

        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)]]
            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))
            mlp_model = MLP(config)

            best_auc = 0
            best_epoch = 0
            best_model = None
            print("start training...")
            for epoch in range(max_epoch):
                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(mlp_model, x, d, y)
                    gradients = tape.gradient(batch_cost, mlp_model.trainable_variables)
                    mlp_model.optimizer.apply_gradients(zip(gradients, mlp_model.trainable_variables))
```

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Section

+ Code + Text

Reconnect

```
                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(mlp_model, test_x, test_d, test_y, config)
            #print('epoch:{e}, validation auc:{l:.6f}'.format(e=epoch+1, l=current_auc))
            #validation_auc.append(current_auc)
            if current_auc > best_auc:
                best_auc = current_auc
                best_epoch = epoch+1
                best_model = mlp_model.get_weights()

        #print('Best model of {k}th fold: at epoch {e}, best model validation loss:{l:.6f}'.format(k=i+1, e=best_epoch, l=best_auc))
        k_fold_training_loss.append(training_loss)
        #k_fold_validation_auc.append(validation_auc)

        print("calculate AUC on the best model using the test set")
        mlp_model.set_weights(best_model)
        test_auc = calculate_auc(mlp_model, test_x, test_d, test_y, config)
        test_ppv = calculate_ppv(mlp_model, test_x, test_d, test_y, config)
        print("AUC of {k}th fold: {auc:.6f}".format(k=i+1, auc=test_auc))
        k_fold_auc.append(test_auc)
        k_fold_ppv.append(test_ppv)
        fpr, tpr, thresholds = calculate_ROC(mlp_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, "MLP_{m}_{k}fold_l{l}_training_loss_ver{i}.npy".format(k=k, m=mode_name, l=learning_rate, i = version)), k_fold_training_loss)
    np.save(os.path.join(output_path, "MLP_{m}_{k}fold_l{l}_auc_ver{i}.npy".format(k=k, m=mode_name, l=learning_rate, i=version)), k_fold_auc)
    np.save(os.path.join(output_path, "MLP_{m}_{k}fold_l{l}_tpr_ver{i}.npy".format(k=k, m=mode_name, l=learning_rate, i=version)), k_fold_tpr)
    np.save(os.path.join(output_path, "MLP_{m}_{k}fold_l{l}_ppv_ver{i}.npy".format(k=k, m=mode_name, l=learning_rate, i=version)), k_fold_ppv)
    #np.save(os.path.join(output_path, "MLP_{m}_{k}fold_l{l}_model_ver{i}.npy".format(k=k, m=mode_name, l=learning_rate, i=version)), mlp_model.get_weights())
  save_data(os.path.join(output_path, "MLP_{m}_{k}fold_l{l}_config.pkl".format(k=k, m=mode_name, l=learning_rate)), config)
train_mlp_kfold('/content/drive/MyDrive/Colab Notebooks/Data/ml_output', '/content/drive/MyDrive/Colab Notebooks/Data/patient_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/demo_records.pkl','/content/drive/MyDrive/Colab Notebooks/Data/labels.pkl', 20, 2,100, 4,hid
```

```
52/52 [==============================] - 1s 25ms/step
epoch:15, mean loss:0.961713
52/52 [==============================] - 2s 30ms/step
epoch:16, mean loss:0.961276
52/52 [==============================] - 1s 25ms/step
epoch:17, mean loss:0.953789
52/52 [==============================] - 1s 19ms/step
epoch:18, mean loss:0.948283
52/52 [==============================] - 1s 19ms/step
epoch:19, mean loss:0.946016
52/52 [==============================] - 1s 18ms/step
epoch:20, mean loss:0.942642
calculate AUC on the best model using the test set
AUC of 4th fold: 0.300000
build and initialize model for 5th fold...
start training...
52/52 [==============================] - 1s 17ms/step
epoch:1, mean loss:1.768654
52/52 [==============================] - 1s 17ms/step
epoch:2, mean loss:1.360882
52/52 [==============================] - 1s 18ms/step
epoch:3, mean loss:1.234782
52/52 [==============================] - 1s 21ms/step
epoch:4, mean loss:1.176759
52/52 [==============================] - 1s 21ms/step
```

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved


```
epoch:20, mean loss:0.942642
calculate AUC on the best model using the test set
AUC of 4th fold: 0.300000
build and initialize model for 5th fold...
start training...
52/52 [==============================] - 1s 17ms/step
epoch:1, mean loss:1.768654
52/52 [==============================] - 1s 17ms/step
epoch:2, mean loss:1.360882
52/52 [==============================] - 1s 18ms/step
epoch:3, mean loss:1.234782
52/52 [==============================] - 1s 21ms/step
epoch:4, mean loss:1.176759
52/52 [==============================] - 1s 21ms/step
epoch:5, mean loss:1.126980
52/52 [==============================] - 1s 21ms/step
epoch:6, mean loss:1.078730
52/52 [==============================] - 1s 25ms/step
epoch:7, mean loss:1.037788
52/52 [==============================] - 1s 28ms/step
epoch:8, mean loss:1.007759
52/52 [==============================] - 1s 25ms/step
epoch:9, mean loss:0.983325
52/52 [==============================] - 1s 22ms/step
epoch:10, mean loss:0.959759
52/52 [==============================] - 1s 19ms/step
epoch:11, mean loss:0.945116
52/52 [==============================] - 1s 20ms/step
epoch:12, mean loss:0.929217
52/52 [==============================] - 1s 19ms/step
epoch:13, mean loss:0.913263
52/52 [==============================] - 1s 20ms/step
epoch:14, mean loss:0.908038
52/52 [==============================] - 1s 21ms/step
epoch:15, mean loss:0.896690
52/52 [==============================] - 1s 18ms/step
epoch:16, mean loss:0.884945
52/52 [==============================] - 1s 18ms/step
epoch:17, mean loss:0.880293
52/52 [==============================] - 1s 18ms/step
epoch:18, mean loss:0.874474
52/52 [==============================] - 1s 18ms/step
epoch:19, mean loss:0.868452
52/52 [==============================] - 1s 22ms/step
epoch:20, mean loss:0.861789
calculate AUC on the best model using the test set
AUC of 5th fold: 0.791667
saving k-fold results...
```

Results

Logistic Regression's Higher AUC might indicate that for this particular dataset and task, the simpler LR model is better at distinguishing between the positive and negative classes. The higher AUC indicates a better overall prediction accuracy at various threshold settings.

MLP's Lower Performance demonstrated that several potential issues such as overfitting to the training data, underfitting due to insufficient training epochs or data, inadequate architecture complexity, or that the hyperparameters were not optimally set for this specific task.

```
# metrics to evaluate my model

# plot figures to better show the results
```

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

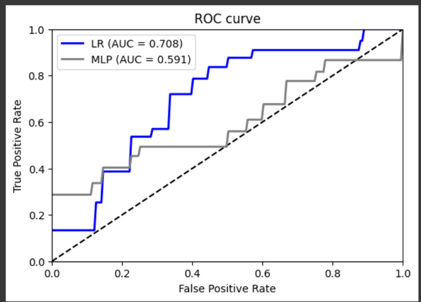
+ Section

+ Code + Text

Reconnect

training epochs or data, inadequate architecture complexity, or that the hyperparameters were not optimally set for this specific task.

```
[ ] # 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)
    tpr_MLP = np.load('/content/drive/MyDrive/Colab Notebooks/Data/ml_output/MLP_mhot_5fold_l0.001_tpr_ver0.npy')
    auc_MLP = np.load('/content/drive/MyDrive/Colab Notebooks/Data/ml_output/MLP_mhot_5fold_l0.001_auc_ver0.npy')
    mean_tpr_MLP = np.mean(tpr_MLP, axis=0)
    mean_auc_MLP = np.mean(auc_MLP, 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.plot(mean_fpr, mean_tpr_MLP, color='gray',label=r'MLP (AUC = %0.3f)' % (mean_auc_MLP),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

Team63_DLH_Project

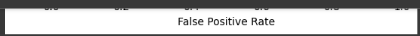
File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Section

+ Code + Text

Reconnect



Model comparison

Paper LR model AUC: 0.87

Our LR model AUC: 0.708

```
[ ] # 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

------------------LR MODEL ------------------

In this final version of the project after the code implementation and training we noticed from the graph below the model is getting a score of 0.708 AUC instead of the 0.87 AUC achieved in the research paper. This is great but not excellent. We believe this difference is due to the dataset limitation as we do not have access to the NYU UCIMC; however I think if we have more time to better tune the hyperparameters there are indeed room for high AUC to improve.

But overall according to this 0.708 AUC it sugguests that the LR 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.

------------------MLP MODEL ------------------

Despite the dataset difference we already mentioned, this model has an AUC of 0.591, which is below the threshold commonly considered as good. This indicates that the MLP's performance in classifying the positive class in the data is relatively poor compared to the LR model. We believe the low AUC might suggest several potential issues such as overfitting to the training data, underfitting due to insufficient training epochs or data, inadequate architecture complexity, or that the hyperparameters were not optimally set for this specific task.

Overall

After the deep learning trained with both models - Logistic Regression and MLP, we did find that the LR model achieved the best performance in identifying delirium with a mean of 0.708. This result does align with paper's result with also demonstrated that LR achieved a higher AUC of 0.874. Their AUC is higher than our project is due to serveral factors

- Dataset availbility
- Dataset training scale
- Hyperparam tunning

This is indeed a very meaningful project which provided us experience with hands on real world application model training that can help us establish a better understanding on how model training can be used in the health care field to predict delirium.

Team63_DLH_Project

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Table of contents

+ Section

+ Code + Text

Reconnect

good. This indicates that the MLP's performance in classifying the positive class in the data is relatively poor compared to the LR model. We believe the low AUC might suggest several potential issues such as overfitting to the training data, underfitting due to insufficient training epochs or data, inadequate architecture complexity, or that the hyperparameters were not optimally set for this specific task.

Overall

After the deep learning trained with both models - Logistic Regression and MLP, we did find that the LR model achieved the best performance in identifying delirium with a mean of 0.708. This result does align with paper's result with also demonstrated that LR achieved a higher AUC of 0.874. Their AUC is higher than our project is due to serveral factors

- Dataset availbility
- Dataset training scale
- Hyperparam tunning

This is indeed a very meaningful project which provided us experience with hands on real world application model training that can help us establish a better understanding on how model training can be used in the health care field to predict delirium.

```
[ ] # 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

Jae Hyun Kim, May Hua, Robert A Whittington, Junghwan Lee, Cong Liu, Casey N Ta, Edward R Marcantonio, Terry E Goldberg, Chunhua Weng, A machine learning approach to identifying delirium from electronic health records, JAMIA Open, Volume 5, Issue 2, July 2022, ooac042, https://doi.org/10.1093/jamiaopen/ooac042

Feel free to add new sections