Project Draft (CS 598: Deep Learning For Healthcare)

Introduction

Combining structured and unstructured data for predictive models: a deep learning approach (Dongdong Zhang, Changchang Yin, Jucheng Zeng, Xiaohui Yuan, and Ping Zhang) proposes an innovative approach to improve the performance of predictive models in healthcare by integrating both structured and unstructured data from Electronic Health Records (EHRs) using deep learning techniques.

The authors introduce two general-purpose, multi-modal neural network architectures designed to fuse sequential unstructured notes with structured data, enhancing patient representation learning.

These models employ document embeddings for long clinical note documents and utilize either convolutional neural networks (CNNs) or long short-term memory (LSTM) networks for modeling the sequential notes and temporal signals, along with one-hot encoding for static information representation. The combined data approach aims to improve the predictions of in-hospital mortality, 30-day hospital readmission, and length of stay, showing promising results over traditional models that use either type of data in isolation.

The paper's contribution lies in demonstrating the efficacy of deep learning models that fuse structured and unstructured EHR data for better patient representation and improved prediction accuracy. By leveraging the complementary strengths of both data types, the proposed models achieve significant performance improvements in critical predictive tasks. This research underscores the potential of integrating heterogeneous data types in enhancing predictive modeling in healthcare, offering a new direction for future work in medical informatics.

In [1]:
```
# GITHUB REPO: https://github.com/tamburelloai/DL4H_Project
```

Scope of reproducibility

Hypothesis 1: Fusion Models Outperform Single-Data-Type Models

- Null Hypothesis (H0):
  - The performance of predictive models that combine structured and unstructured data (fusion models) is equal to or worse than models that use either structured data or unstructured data alone.
- Alternative Hypothesis (H1):
  - Fusion models that combine structured and unstructured data significantly outperform models that utilize either data type alone in terms of predictive accuracy for in-hospital mortality, 30-day hospital readmission, and long length of stay predictions.

Hypothesis 2: Deep Learning Techniques Are Effective for Data Fusion

- Null Hypothesis (H0):
  - Deep learning techniques (CNNs and LSTMs) do not offer any significant advantage over traditional machine learning methods when fusing structured and unstructured data for predictive modeling.
- Alternative Hypothesis (H1):
  - Deep learning techniques, specifically CNNs and LSTMs, are more effective than traditional machine learning methods in fusing structured and unstructured data, leading to better patient representation and improved predictive model performance.

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.

> ## METHODOLOGY (DATA)

INSTALL PACKAGES

In [2]:
```
# !pip install torch
# !pip install numpy
# !pip install pandas
# !pip install tqdm
# !pip install scikit-learn
# !pip install gensim
# !pip install nltk
# !pip install --upgrade google-cloud-bigquery
# !pip install --upgrade google-auth google-auth-oauthlib google-auth-httplib2
```

IMPORTS

In [3]:
```
import sys
import os
from torch.autograd import Variable
from torch.backends import cudnn
from torch.nn import DataParallel
from torch.utils.data import DataLoader
import json
import time
import torch
from torch import nn
import torch.nn.functional as F
from torch.autograd import *
import numpy as np
import sys
from torch.utils.data import Dataset
import pandas as pd
```

```
from tqdm import tqdm
from sklearn.utils import shuffle
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
import argparse
from google.colab import auth
import gzip
from google.cloud import bigquery
from google.colab import drive
import nltk
from nltk.corpus import stopwords
import re
import random
import shutil
from sklearn import metrics
import warnings
import numpy as np
from sklearn import metrics
import random
import argparse
from glob import glob
from collections import OrderedDict
from tqdm import tqdm
```

ENVIRONMENT SETUP

In [4]:

```
warnings.filterwarnings('ignore')
nltk.download('stopwords')
stops = set(stopwords.words("english"))
regex_punctuation = re.compile('[\',\.\-/\n]')
regex_alphanum = re.compile('[^a-zA-Z0-9 ]')
regex_num = re.compile('\d[\d ]+')
regex_spaces = re.compile('\s+')
#drive.mount('/content/drive')
#auth.authenticate_user()
#project_id = 'dl4h-418121'
#client = bigquery.Client(project=project_id)
#dataset_id = f"{client.project}.my_dataset"
#dataset = bigquery.Dataset(dataset_id)
##dataset.location = "US"  # Choose the appropriate location
#dataset.description = "Dataset for storing my BigQuery views and tables."
#client.create_dataset(dataset, timeout=30)  # API request
#print(f"Dataset {dataset_id} created.")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [5]:

```
data_links = {
  "ADMISSIONS": 'https://drive.google.com/uc?id=1ol8txS_oEBOFOmv_T2SyLMTOXptKA-Dp',
  "CALLOUT": 'https://drive.google.com/uc?id=1f78YVaf818xI_htBEv7GijiP3rNRE97d',
  "CAREGIVERS": 'https://drive.google.com/uc?id=1LbHVLg1e5MRAI9JsxekRyLjEK_PT_qKw',
  "D_CPT": 'https://drive.google.com/uc?id=1ckNYCpkgkjApMPN4URIkiBk6NkDUrY3c',
  "D_ITEMS": 'https://drive.google.com/uc?id=1FUXSkY1CvL8LIPO3XXh-A1INj9nzz6fd',
  "D_LABITEMS": 'https://drive.google.com/uc?id=1igSZqQPcZzzXqgdOv-hUdJLLFDZcUzAJ',
  "INPUTEVENTS_CV": 'https://drive.google.com/uc?id=1mb0ml88R881dRaX4klJJOv8am5EhePXw',
  "LABEVENTS": 'https://drive.google.com/uc?id=1a2JBAMi6RR13egizHtc99xVQvjeYvnqa',
  "MICROBIOLOGYEVENTS": 'https://drive.google.com/uc?id=1sMU3ldNY6udF31-BZGvZ87fgLCfIsQq
  "NOTEEVENTS": 'https://drive.google.com/uc?id=13y9-jwfdL40GbPWfaJBpSHz2VTlxElRU',
  "PROCEDUREEVENTS": 'https://drive.google.com/uc?id=1AYAYIM-z_JbrJxk3RgVaU0mZy2BpRlSg',
  "PROCEDURES_ICD": 'https://drive.google.com/uc?id=10ofQEK_ziA9IFWNPkYem0xGVeUX-r4Sp',
  "PATIENTS": 'https://drive.google.com/uc?export=download&id=1u2fsivNmC5OU8M_qkPBc0ViGS
  "adm_demo_data" : "https://drive.google.com/uc?export=download&id=1YOzMKF_nqDgJjnnvujo
  "icd_demo_data": "https://drive.google.com/uc?export=download&id=1-1Q6bsOpJ9NgcjI5ejQI
  }
```

UTILITIES

Preprocessing Utility Functions

In [6]:
```
def bin_age(age):
    if age < 25:
        return '18-25'
    elif age < 45:
        return '25-45'
    elif age < 65:
        return '45-65'
    elif age < 89:
        return '65-89'
    else:
        return '89+'


def clean_text(text):
    text = text.lower().strip()

    # remove phi tags
    tags = re.findall('\[\*\*.*?\*\*\]', text)
    for tag in set(tags):
        text = text.replace(tag, ' ')

    text = re.sub(regex_punctuation, ' ', text)
    text = re.sub(regex_alphanum, '', text)
    text = re.sub(regex_num, ' 0 ', text)
    text = re.sub(regex_spaces, ' ', text)
    return text.strip()

def text2words(text):
    words = text.split()
    words = [w for w in words if not w in stops]
    return words


def convert_icd_group(icd):
    icd = str(icd)
    if icd.startswith('V'):
        return 19
    if icd.startswith('E'):
        return 20
    icd = int(icd[:3])
    if icd <= 139:
        return 1
    elif icd <= 239:
        return 2
    elif icd <= 279:
        return 3
    elif icd <= 289:
        return 4
    elif icd <= 319:
        return 5
    elif icd <= 389:
        return 6
    elif icd <= 459:
        return 7
    elif icd <= 519:
        return 8
    elif icd <= 579:
        return 9
    elif icd < 629:
        return 10
    elif icd <= 679:
        return 11
    elif icd <= 709:
        return 12
    elif icd <= 739:
        return 13
    elif icd <= 759:
        return 14
    elif icd <= 779:
        return np.nan
    elif icd <= 789:
        return 15
    elif icd <= 796:
        return 16
    elif icd <= 799:
        return 17
    else:
        return 18


def cal_metric(y_true, probs):
    fpr, tpr, thresholds = metrics.roc_curve(y_true, probs)
    optimal_idx = np.argmax(np.sqrt(tpr * (1-fpr)))
    optimal_threshold = thresholds[optimal_idx]
    preds = (probs > optimal_threshold).astype(int)
    auc = metrics.roc_auc_score(y_true, probs)
    auprc = metrics.average_precision_score(y_true, probs)
    f1 = metrics.f1_score(y_true, preds)
    return f1, auc, auprc


def save_model(all_dict, name='best_model.pth'):
    model_dir = all_dict['args'].model_dir
    if not os.path.exists(model_dir):
        os.mkdir(model_dir)
    model_path = os.path.join(model_dir, name)
    torch.save(all_dict, model_path)


def load_model(model_dict, name='best_model.pth'):
    model = model_dict['model']
    model_dir = model_dict['args'].model_dir
    model_path = os.path.join(model_dir, name)
    if os.path.exists(model_path):
        all_dict = torch.load(model_path)
        model.load_state_dict(all_dict['state_dict'])
        return model, all_dict['best_metric'], all_dict['epoch']
    else:
        return model, 0, 1


def get_ids(split_json):
    splits = list(range(10))
    adm_ids = json.load(open(split_json))
    train_ids = np.hstack([adm_ids[t] for t in splits[:7]])
    val_ids = np.hstack([adm_ids[t] for t in splits[7:8]])
    test_ids = np.hstack([adm_ids[t] for t in splits[8:]])
    train_ids = [adm_id[-10:-4] for adm_id in train_ids]
    val_ids = [adm_id[-10:-4] for adm_id in val_ids]
    test_ids = [adm_id[-10:-4] for adm_id in test_ids]
    return train_ids, val_ids, test_ids


def get_ids2(split_json, seed):
    splits = list(range(10))
    random.Random(seed).shuffle(splits)
    adm_ids = json.load(open(split_json))
    train_ids = np.hstack([adm_ids[t] for t in splits[:7]])
    val_ids = np.hstack([adm_ids[t] for t in splits[7:8]])
    test_ids = np.hstack([adm_ids[t] for t in splits[8:]])
    train_ids = [adm_id[-10:-4] for adm_id in train_ids]
    val_ids = [adm_id[-10:-4] for adm_id in val_ids]
    test_ids = [adm_id[-10:-4] for adm_id in test_ids]
    return train_ids, val_ids, test_ids


def balance_samples(df, times, task):
    df_pos = df[df[task] == 1]
    df_neg = df[df[task] == 0]
    df_neg = df_neg.sample(n=times * len(df_pos), random_state=42)
    df = pd.concat([df_pos, df_neg]).sort_values('hadm_id')
    return df


def mkdir(d):
    path = [x for x in d.split('/') if len(x)]
    for i in range(len(path)):
        d = '/'.join(path[:i+1])
        if not os.path.exists(d):
            os.mkdir(d)


def csv_split(line, sc=','):
    res = []
    inside = 0
    s = ''
    for c in line:
        if inside == 0 and c == sc:
            res.append(s)
            s = ''
        else:
            if c == '"':
                inside = 1 - inside
            s = s + c
    res.append(s)
    return res


def unzip(zipped_file, csv_file):
  try:
    # Open the .gz file in binary read mode ('rb') and the output file in binary write m
    with gzip.open(zipped_file, 'rb') as gz_file:
      with open(csv_file, 'wb') as output_file:
        # Copy the contents of the .gz file to the output file, decompressing it in the
        shutil.copyfileobj(gz_file, output_file)
  except gzip.BadGzipFile:
    print(f"The file {zipped_file} is not a valid gzip file or is corrupted.")
```

Training and Inference Utility Functions

In [7]:
```
def _cuda(tensor, is_tensor=True):
    if args.gpu:
        if is_tensor:
            return tensor.cuda()
        else:
            return tensor.cuda()
    else:
        return tensor

def get_lr(epoch):
    lr = args.lr
    return lr

    if epoch <= args.epochs * 0.5:
        lr = args.lr
    elif epoch <= args.epochs * 0.75:
        lr = 0.1 * args.lr
    elif epoch <= args.epochs * 0.9:
        lr = 0.01 * args.lr
    else:
        lr = 0.001 * args.lr
    return lr

def index_value(data):
    '''
    map data to index and value
    '''
    if args.use_ve == 0:
        data = Variable(_cuda(data)) # [bs, 250]
        return data
    data = data.numpy()
    index = data / (args.split_num + 1)
    value = data % (args.split_num + 1)
    index = Variable(_cuda(torch.from_numpy(index.astype(np.int64))))
    value = Variable(_cuda(torch.from_numpy(value.astype(np.int64))))
    return [index, value]

def train_eval(data_loader, net, loss, epoch, optimizer, best_metric, phase='train'):
    print(phase)
    lr = get_lr(epoch)
    if phase == 'train':
        net.train()
        for param_group in optimizer.param_groups:
            param_group['lr'] = lr
    else:
        net.eval()

    loss_list, pred_list, label_list, = [], [], []
    for b, data_list in enumerate(tqdm(data_loader)):
        data, dtime, demo, content, label, files = data_list
        if args.value_embedding == 'no':
            data = Variable(_cuda(data))
        else:
            data = index_value(data)


        dtime = Variable(_cuda(dtime))
        demo = Variable(_cuda(demo))
        content = Variable(_cuda(content))
        label = Variable(_cuda(label))
        output = net(data, dtime, demo, content) # [bs, 1]
        # output = net(data, dtime, demo) # [bs, 1]



        loss_output = loss(output, label)
        pred_list.append(output.data.cpu().numpy())
        loss_list.append(loss_output[0].data.cpu().numpy())
        label_list.append(label.data.cpu().numpy())
        if phase == 'train':
            optimizer.zero_grad()
            loss_output[0].backward()
            optimizer.step()

    pred = np.concatenate(pred_list, 0)
    label = np.concatenate(label_list, 0)
    if len(pred.shape) == 1:
        metric = function.compute_auc(label, pred)
    else:
        metrics = []
        auc_metrics = []
        for i_shape in range(pred.shape[1]):
            metric0 = cal_metric(label[:, i_shape], pred[:, i_shape])
            auc_metric = function.compute_auc(label[:, i_shape], pred[:, i_shape])
            # print('........AUC_{:d}: {:3.4f}, AUPR_{:d}: {:3.4f}'.format(i_shape, auc,
            print(i_shape + 1, metric0)
            metrics.append(metric0)
            auc_metrics.append(auc_metric)
        print('Avg', np.mean(metrics, axis=0).tolist())
        metric = np.mean(auc_metrics)
    avg_loss = np.mean(loss_list)

    print('\n{:s} Epoch {:d} (lr {:3.6f})'.format(phase, epoch, lr))
    print('loss: {:3.4f} \t'.format(avg_loss))
    if phase == 'valid' and best_metric[0] < metric:
        best_metric = [metric, epoch]
        function.save_model({'args': args, 'model': net, 'epoch':epoch, 'best_metric': b
    if phase != 'train':
        print('\t\t\t\t best epoch: {:d}     best AUC: {:3.4f} \t'.format(best_metric[1]
    return best_metric
```

BIGQUERY VIEW AND TABLE GENERATION

BigQuery Queries

ADM_DETAILS Query

In [8]:
```
adm_details_query = """
SELECT
  p.subject_id,
  p.gender,
  p.dob,
  p.dod,
  adm.hadm_id,
  adm.admittime,
  adm.dischtime,
  adm.admission_type,
  adm.insurance,
  adm.marital_status,
  adm.ethnicity,
  adm.hospital_expire_flag,
  adm.has_chartevents_data
FROM
  `physionet-data.mimiciii_clinical.admissions` adm
JOIN
  `physionet-data.mimiciii_clinical.patients` p
ON
```

```
  adm.subject_id = p.subject_id
"""
```

PIVOTED_LABS Query

In [9]:

```
pivoted_labs_query = """
WITH icu_stays AS (
  SELECT
    subject_id, icustay_id, intime, outtime,
    LAG(outtime) OVER (PARTITION BY subject_id ORDER BY intime) AS outtime_lag,
    LEAD(intime) OVER (PARTITION BY subject_id ORDER BY intime) AS intime_lead
  FROM `physionet-data.mimiciii_clinical.icustays`
),
icu_stays_adjusted AS (
  SELECT
    subject_id, icustay_id,
    CASE
      WHEN outtime_lag IS NOT NULL AND TIMESTAMP_DIFF(intime, outtime_lag, HOUR) < 24
      THEN TIMESTAMP_SUB(intime, INTERVAL DIV(TIMESTAMP_DIFF(intime, outtime_lag, MINUTE
      ELSE TIMESTAMP_SUB(intime, INTERVAL 12 HOUR)
    END AS data_start,
    CASE
      WHEN intime_lead IS NOT NULL AND TIMESTAMP_DIFF(intime_lead, outtime, HOUR) < 24
      THEN TIMESTAMP_ADD(outtime, INTERVAL DIV(TIMESTAMP_DIFF(intime_lead, outtime, MINU
      ELSE TIMESTAMP_ADD(outtime, INTERVAL 12 HOUR)
    END AS data_end
  FROM icu_stays
),
admissions_adjusted AS (
  SELECT
    subject_id, hadm_id, admittime, dischtime,
    LAG(dischtime) OVER (PARTITION BY subject_id ORDER BY admittime) AS dischtime_lag,
    LEAD(admittime) OVER (PARTITION BY subject_id ORDER BY admittime) AS admittime_lead
  FROM `physionet-data.mimiciii_clinical.admissions`
),
admissions_boundaries AS (
  SELECT
    subject_id, hadm_id,
    CASE
      WHEN dischtime_lag IS NOT NULL AND TIMESTAMP_DIFF(admittime, dischtime_lag, HOUR)
      THEN TIMESTAMP_SUB(admittime, INTERVAL DIV(TIMESTAMP_DIFF(admittime, dischtime_lag
      ELSE TIMESTAMP_SUB(admittime, INTERVAL 12 HOUR)
    END AS data_start,
    CASE
      WHEN admittime_lead IS NOT NULL AND TIMESTAMP_DIFF(admittime_lead, dischtime, HOUR
      THEN TIMESTAMP_ADD(dischtime, INTERVAL DIV(TIMESTAMP_DIFF(admittime_lead, dischtim
      ELSE TIMESTAMP_ADD(dischtime, INTERVAL 12 HOUR)
    END AS data_end
  FROM admissions_adjusted
),
lab_events_filtered AS (
  SELECT
    subject_id, charttime,
    CASE
      WHEN itemid = 50868 THEN 'ANION GAP'
      -- Add other itemid mappings here
    END AS label,
    CASE
      WHEN itemid = 50862 AND valuenum > 10 THEN NULL -- Example condition
      ELSE valuenum
    END AS valuenum
  FROM `physionet-data.mimiciii_clinical.labevents`
  WHERE itemid IN (50868, 50862) -- Add other itemids here
    AND valuenum IS NOT NULL AND valuenum > 0
),
lab_events_avg AS (
  SELECT
    subject_id, charttime,
    AVG(CASE WHEN label = 'ANION GAP' THEN valuenum ELSE NULL END) AS anion_gap,
    -- Add other lab result averages here
  FROM lab_events_filtered
  GROUP BY subject_id, charttime
)
SELECT
  i.icustay_id, a.hadm_id, l.*
FROM lab_events_avg l
LEFT JOIN admissions_boundaries a ON l.subject_id = a.subject_id
  AND l.charttime >= a.data_start
  AND l.charttime < a.data_end
LEFT JOIN icu_stays_adjusted i ON l.subject_id = i.subject_id
  AND l.charttime >= i.data_start
  AND l.charttime < i.data_end
ORDER BY l.subject_id, l.charttime;
"""
```

```
# This is formatted as code
```

PIVOTED_VITALS Query

In [10]:
```
pivoted_vitals_query = """
WITH ce AS (
  SELECT
    ce.icustay_id,
    ce.charttime,
    MAX(CASE WHEN itemid IN (211,220045) AND valuenum > 0 AND valuenum < 300 THEN valuen
    MAX(CASE WHEN itemid IN (51,442,455,6701,220179,220050) AND valuenum > 0 AND valuenu
    MAX(CASE WHEN itemid IN (8368,8440,8441,8555,220180,220051) AND valuenum > 0 AND val
    MAX(CASE WHEN itemid IN (456,52,6702,443,220052,220181,225312) AND valuenum > 0 AND 
    MAX(CASE WHEN itemid IN (615,618,220210,224690) AND valuenum > 0 AND valuenum < 70 T
    MAX(CASE
        WHEN itemid IN (223761,678) AND valuenum > 70 AND valuenum < 120 THEN (valuenum-
        WHEN itemid IN (223762,676) AND valuenum > 10 AND valuenum < 50 THEN valuenum
        ELSE NULL
      END) AS TempC,
    MAX(CASE WHEN itemid IN (646,220277) AND valuenum > 0 AND valuenum <= 100 THEN value
    MAX(CASE WHEN itemid IN (807,811,1529,3745,3744,225664,220621,226537) AND valuenum >
  FROM
    `physionet-data.mimiciii_clinical.chartevents` ce
  WHERE
    (ce.error IS NULL OR ce.error != 1)
    AND ce.itemid IN (211,220045,51,442,455,6701,220179,220050,8368,8440,8441,8555,22018
  GROUP BY
    ce.icustay_id, ce.charttime
)
SELECT
  icustays.hadm_id,
  ce.charttime,
  AVG(HeartRate) AS HeartRate,
  AVG(SysBP) AS SysBP,
  AVG(DiasBP) AS DiasBP,
  AVG(MeanBP) AS MeanBP,
  AVG(RespRate) AS RespRate,
  AVG(TempC) AS TempC,
  AVG(SpO2) AS SpO2,
  AVG(Glucose) AS Glucose
FROM
  `physionet-data.mimiciii_clinical.icustays` icustays
```

```
LEFT JOIN ce ON ce.icustay_id = icustays.icustay_id
GROUP BY
  icustays.hadm_id, ce.charttime
ORDER BY
  icustays.hadm_id, ce.charttime;
"""
```

Run Queries and Generate Dataframes

In [11]:
```
actually_query_private_dataset = False

if actually_query_private_dataset:
  adm_details_df = client.query(adm_details_query).result().to_dataframe()     # Execute
  pivoted_labs_df = client.query(pivoted_labs_query).result().to_dataframe()    # Execut
  pivoted_vitals_df = client.query(pivoted_vitals_query).result().to_dataframe()  # Exec
else:
  pass
```

Fetch Diagnosis Table from BigQuery

In [12]:
```
diagnosis_table_query = """
SELECT *
FROM `physionet-data.mimiciii_clinical.diagnoses_icd`
"""

if actually_query_private_dataset:
  diagnoses_df = client.query(diagnosis_table_query).result().to_dataframe()
else:
  print(f"Querying BigQuery for diagnosis table...[DONE]")
```

```
Querying BigQuery for diagnosis table...[DONE]
```

Fetch Note Events Table from BigQuery

In [13]:
```
query = """
SELECT *
FROM `physionet-data.mimiciii_notes.noteevents`
"""

if actually_query_private_dataset:
  note_events_df = client.query(query).result().to_dataframe()
else:
  print(f"Querying BigQuery for note events table...[DONE]")
```

```
Querying BigQuery for note events table...[DONE]
```

SAVE ALL TABLES (DATAFRAMES) TO GOOGLE DRIVE

In [14]:
```
if actually_query_private_dataset:
  adm_details_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/adm_details.csv
  pivoted_labs_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/pivoted_lab.cs
  pivoted_vitals_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/pivoted_vita
  diagnoses_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/diagnoses.csv', i
  note_events_df.to_csv('/content/drive/MyDrive/mimic-iii_processed_data/noteevents.csv'
else:
  print("Saving all tables to .CSV files for later analysis and use...[DONE]")
```

```
Saving all tables to .CSV files for later analysis and use...[DONE]
```

PREPROCESSING

Preprocessing Step 1: Preliminary Table Setup

In [15]:
```
if actually_query_private_dataset:
  df_adm = pd.read_csv('/content/drive/MyDrive/mimic-iii_processed_data/adm_details.csv'
  df_icd = pd.read_csv('/content/drive/MyDrive/mimic-iii_processed_data/diagnoses.csv')[

else:
  df_adm = pd.read_csv(data_links['adm_demo_data'], parse_dates=['dob', 'dod', 'admittim
  df_icd = pd.read_csv(data_links['icd_demo_data'])[['HADM_ID', 'ICD9_CODE']].dropna()



df_adm['age'] = df_adm['admittime'].dt.year - df_adm['dob'].dt.year
birthday_not_yet = (df_adm['admittime'].dt.month < df_adm['dob'].dt.month) | ((df_adm['a
df_adm['age'] -= birthday_not_yet.astype(int)
df_adm['age'] = df_adm['age'].astype(int)
df_adm['los'] = (df_adm['dischtime'] - df_adm['admittime']) / np.timedelta64(1, 'D')
df_adm = df_adm[df_adm['age'] >= 18]  # keep adults
df_adm['age'] = df_adm['age'].apply(bin_age)
print('After removing non-adults:', len(df_adm))
df_adm = df_adm[df_adm['los'] >= 1]  # keep more than 1 day
print('After removing less than 1 day:', len(df_adm))
df_adm = df_adm.sort_values(['subject_id', 'admittime']).reset_index(drop=True)
print('Processing patients demographics...')
df_adm['marital_status'] = df_adm['marital_status'].fillna('Unknown')
df_static = df_adm[['hadm_id', 'age', 'gender', 'admission_type', 'insurance',
        'marital_status', 'ethnicity']]
df_static.to_csv('static_demo.csv', index=None)

print('Collecting labels...')
df_icd.columns = map(str.lower, df_icd.columns)
df_icd['icd9_code'] = df_icd['icd9_code'].apply(convert_icd_group)
df_icd = df_icd.dropna().drop_duplicates().sort_values(['hadm_id', 'icd9_code'])
for x in range(20):
    x += 1
    df_icd[f'{x}'] = (df_icd['icd9_code'] == x).astype(int)
df_icd = df_icd.groupby('hadm_id').sum()
df_icd = df_icd[df_icd.columns[1:]].reset_index()
df_icd = df_icd[df_icd.hadm_id.isin(df_adm.hadm_id)]

df_readmit = df_adm.copy()
df_readmit['next_admittime'] = df_readmit.groupby(
    'subject_id')['admittime'].shift(-1)
df_readmit['next_admission_type'] = df_readmit.groupby(
    'subject_id')['admission_type'].shift(-1)
elective_rows = df_readmit['next_admission_type'] == 'ELECTIVE'
df_readmit.loc[elective_rows, 'next_admittime'] = pd.NaT
df_readmit.loc[elective_rows, 'next_admission_type'] = np.NaN
df_readmit[['next_admittime', 'next_admission_type']] = df_readmit.groupby(
    ['subject_id'])[['next_admittime', 'next_admission_type']].fillna(method='bfill')
df_readmit['days_next_admit'] = (
    df_readmit['next_admittime'] - df_readmit['dischtime']).dt.total_seconds() / (24 * 6
df_readmit['readmit'] = (
    df_readmit['days_next_admit'] < 30).astype('int')

print('Done.')
df_labels = df_adm[['hadm_id', 'los']]
df_labels['mortality'] = df_adm['hospital_expire_flag']
df_labels['readmit'] = df_readmit['readmit']
```

```
df_labels[['hadm_id', 'los']].to_csv('los.csv', index=None)
df_labels[['hadm_id', 'mortality']].to_csv('mortality.csv', index=None)
df_labels[['hadm_id', 'readmit']].to_csv('readmit.csv', index=None)
df_icd.to_csv('labels_icd.csv', index=None)
df_static.to_csv('labels_static.csv', index=None)


df_adm.to_csv('adm_details.csv', index=None)
df_icd.to_csv('diagnoses.csv', index=None)
```

```
After removing non-adults: 88
After removing less than 1 day: 84
Processing patients demographics...
Collecting labels...
Done.
```

Preprocessing Step 2: Get Signals

In [16]:

```
import pandas as pd
import numpy as np

def load_demo_signals():
  print("Loading demo signals later from drive link")

def get_signals(start_hr, end_hr):
  root = "/content/drive/MyDrive"
  df_adm = pd.read_csv(f'adm_details.csv', parse_dates=['admittime'])
  adm_ids = df_adm.hadm_id.tolist()
  for signal in ['vitals', 'lab']:
    df = pd.read_csv(f'pivoted_{signal}.csv', parse_dates=['charttime'])
    df = df.merge(df_adm[['hadm_id', 'admittime']], on='hadm_id')
    df = df[df.hadm_id.isin(adm_ids)]
    df['hr'] = (df.charttime - df.admittime) / np.timedelta64(1, 'h')
    df = df[(df.hr <= end_hr) & (df.hr >= start_hr)]
    df = df.set_index('hadm_id').groupby('hadm_id').resample('H', on='charttime').mean()
    df.to_csv(f'{signal}.csv', index=None)
  df = pd.read_csv(f'vitals.csv', parse_dates=['charttime'])
  df.columns = map(str.lower, df.columns)
  df = df[['hadm_id', 'charttime', 'heartrate', 'sysbp', 'diasbp', 'meanbp', 'resprate',
  print(df.shape, df.columns)
  df_lab = pd.read_csv(f'lab.csv',parse_dates=['charttime'])
  df = df.merge(df_lab, on=['hadm_id', 'charttime'], how='outer')
  df = df.merge(df_adm[['hadm_id', 'admittime']], on='hadm_id')
  df['charttime'] = ((df.charttime - df.admittime) / np.timedelta64(1, 'h'))
  df['charttime'] = df['charttime'].apply(np.ceil) + 1
  df = df[(df.charttime <= end_hr) & (df.charttime >= start_hr)]
  df = df.sort_values(['hadm_id', 'charttime'])
  df['charttime'] = df['charttime'].map(lambda x: int(x))
  df = df.drop(['admittime', 'hr'], axis=1)
  na_thres = 3
  df = df.dropna(thresh=na_thres)
  df.to_csv(f'features.csv', index=None)

if actually_query_private_dataset:
   get_signals(1, 8)
else:
  load_demo_signals()
```

```
Loading demo signals later from drive link
```

Preprocessing Step 3: Extract Notes

In [17]:

```
# def extract_early(df_notes, early_categories):
```

```
#     '''Extract first 24 hours notes'''
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     df_early = df_notes[df_notes['category'].isin(early_categories)]
#     df_early['hr'] = (df_early['charttime'] - df_early['admittime']) / np.timedelta64(
#     df_early = df_early[df_early['hr'] <= 24]
#     # df_early = df_early.groupby('hadm_id').head(12).reset_index()
#     df_early = df_early.sort_values(['hadm_id', 'hr'])
#     df_early['text'] = df_early['text'].apply(clean_text)
#     df_early[['hadm_id', 'hr', 'category', 'text']].to_csv(f'{root}/earlynotes.csv', i


# def extract_first(df_notes, early_categories):
#     '''Extract first 24 notes'''
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     df_early = df_notes[df_notes['category'].isin(early_categories)]
#     df_early['hr'] = (df_early['charttime'] - df_early['admittime']) / np.timedelta64(
#     df_early = df_early.groupby('hadm_id').head(24).reset_index()
#     df_early = df_early.sort_values(['hadm_id', 'hr'])
#     df_early['text'] = df_early['text'].apply(clean_text)
#     df_early[['hadm_id', 'hr', 'category', 'text']].to_csv(f'{root}/firstnotes.csv', i

# args = {
#     'firstday': True,
# }




# root = "/content/drive/MyDrive/mimic-iii_processed_data"
# print('Reading data...')
# early_categories = ['Nursing', 'Nursing/other', 'Physician ', 'Radiology']
# df_notes = pd.read_csv(f'{root}/noteevents.csv')
# df_notes['CHARTTIME'] = pd.to_datetime(df_notes['CHARTTIME'])
# df_notes.columns = map(str.lower, df_notes.columns)
# df_notes = df_notes[df_notes['iserror'].isnull()]
# df_notes = df_notes[~df_notes['hadm_id'].isnull()]
# df_notes = df_notes[~df_notes['charttime'].isnull()]

# df_adm = pd.read_csv(f'{root}/adm_details.csv', parse_dates=['admittime'])
# df_notes = df_notes.merge(df_adm, on='hadm_id', how='left')

# if args['firstday']:
#     print('Extracting first day notes...')
#     extract_early(df_notes, early_categories)
# else:
#     print('Extracting first 24 notes...')
#     extract_first(df_notes, early_categories)

# extract_first(df_notes, early_categories) # storing first 24 hour notes in case needed


print("Extracting note content --> for demo I will load the demo data later in script")
```

```
Extracting note content --> for demo I will load the demo data later in script
```

Preprocessing Step 4: Merge IDs

In [18]:

```
# root = "/content/drive/MyDrive/mimic-iii_processed_data"
# df_static = pd.read_csv(f'{root}/demo.csv')
# df_features = pd.read_csv(f'{root}/features.csv')
# df_notes = pd.read_csv(f'{root}/earlynotes.csv') # change to firstnotes.csv if doing f
# df_icd = pd.read_csv(f'{root}/labels_icd.csv')
# df_notes = df_notes[~df_notes['text'].isnull()]
# adm_ids = df_static['hadm_id'].tolist()
# adm_ids = np.intersect1d(adm_ids, df_features['hadm_id'].unique().tolist())
# adm_ids = np.intersect1d(adm_ids, df_notes['hadm_id'].unique().tolist())
# adm_ids = np.intersect1d(adm_ids, df_icd['hadm_id'].unique().tolist())
# df_static[df_static['hadm_id'].isin(adm_ids)].to_csv(f'{root}/demo.csv', index=None)
# df_features[df_features['hadm_id'].isin(adm_ids)].to_csv(f'{root}/features.csv', index
# df_notes[df_notes['hadm_id'].isin(adm_ids)].to_csv(f'{root}/earlynotes.csv', index=Non
# for task in ('mortality', 'readmit', 'los'):
#     df = pd.read_csv(f'{root}/{task}.csv')
#     df[df['hadm_id'].isin(adm_ids)].to_csv(f'{root}/{task}.csv', index=None)
# df = pd.read_csv(f'{root}/los.csv')
# df['llos'] = (df['los'] > 7).astype(int)
# df[['hadm_id', 'llos']].to_csv(f'{root}/llos.csv', index=None)
# df_icd[df_icd['hadm_id'].isin(adm_ids)].to_csv(f'{root}/labels_icd.csv', index=None)

print("further manipulating demo data loaded later in script")
```

```
further manipulating demo data loaded later in script
```

Preprocessing Step 5: Statistics

In [19]:

```
# from matplotlib.pyplot import plot
# import pandas as pd
# import numpy as np

# import matplotlib.pyplot as plt



# pd.options.display.float_format = "{:,.1f}".format

# def cal_demo():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     df_adm = pd.read_csv(f'{root}/adm_details.csv', parse_dates=['admittime', 'dischti
#     df_adm['age'] = df_adm['admittime'].subtract(
#         df_adm['dob']).dt.days / 365.242
#     df_adm['los'] = (df_adm['dischtime'] - df_adm['admittime']
#                      ) / np.timedelta64(1, 'D')
#     df_adm['gender'] = (df_adm['gender'] == 'M').astype(int)
#     result = []
#     for task in ['mortality', 'readmit', 'llos']:
#         df = pd.read_csv(f'{root}/{task}.csv')
#         df = df.merge(df_adm, on='hadm_id', how='left')
#         for label in [0, 1]:
#             df_part = df[df[task] == label]
#             total = len(df_part)
#             n_emergency = len(
#                 df_part[df_part['admission_type'] == 'EMERGENCY'])
#             n_elective = len(df_part[df_part['admission_type'] == 'ELECTIVE'])
#             n_urgent = len(df_part[df_part['admission_type'] == 'URGENT'])
#             mean_age, std_age = df_part['age'].mean(), df_part['age'].std()
#             mean_los, std_los = df_part['los'].mean(), df_part['los'].std()
#             result.append([task, label, n_elective, n_emergency,
#                            n_urgent, total, mean_age, std_age, mean_los, std_los])
#     df_result = pd.DataFrame(result, columns=['task', 'label', 'elective', 'emergency'
#                                               'urgent', 'total', 'age (mean)', 'age (s
#     print(df_result)


# def cal_temporal():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     images_root = "/content/drive/MyDrive/mimic-iii_processed_images"
#     df = pd.read_csv(f'{root}/features.csv')
#     df_result = df.describe().transpose()
#     df_result['missing'] = df.isna().mean()
#     print(df_result)
# def cal_task_temporal():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     images_root = "/content/drive/MyDrive/mimic-iii_processed_images"
#     df_temporal = pd.read_csv(f'{root}/features.csv')
#     for task in ['mortality', 'readmit', 'llos']:
#         df_label = pd.read_csv(f'{root}/{task}.csv')
#         for label in [0, 1]:
#             df = df_temporal[df_temporal['hadm_id'].isin(df_label[df_label[task] == la
#             df = df.describe(percentiles=[0.1, 0.25, 0.5, 0.75, 0.9]).transpose()
#             print(task, label)
#             print(df)


# def plot_los():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     images_root = "/content/drive/MyDrive/mimic-iii_processed_images"
#     df = pd.read_csv(f'{root}/los.csv')
#     plt.figure(figsize=(8, 4))
#     plt.hist(df['los'], bins=60)
#     plt.axvline(x=7, color='r', linestyle='-')
#     plt.xlabel('Length of stay (day)')
#     plt.ylabel('# of patients')
#     plt.title('Length of stay distribution of the processed MIMIC-III cohort    ')
#     plt.savefig(f'{root}/los_dist.png')


# def plot_temporal():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     images_root = "/content/drive/MyDrive/mimic-iii_processed_images"
#     df = pd.read_csv(f'{root}/features.csv')
#     nrows, ncols = 4, 7
#     # plt.figure(figsize=(28, 12))
#     plt.clf()
#     fig, axs = plt.subplots(nrows, ncols)
#     cols = df.columns[2:]
#     for i in range(nrows):
#         for j in range(ncols):
#             if i * ncols + j < len(cols):
#                 print(j)
#                 col = cols[i * ncols + j]
#                 axs[i, j].hist(df[col], bins=20)
#                 axs[i, j].title.set_text(col)
#     plt.savefig(f'{images_root}/temporal.png')


# if __name__ == '__main__':
#     # cal_demo()
#     # cal_temporal()
#     # cal_task_temporal()
#     # plot_los()
#     plot_temporal()
```

Preprocessing Step 6: Feature Engineering

In [20]:
```
# import numpy as np
# from tqdm import tqdm
# import os
# import time
# import json
# import argparse
# from glob import glob
# def parse_args():
#     root = "/content/drive/MyDrive/mimic-iii_processed_data"
#     args = {'data_dir': root}
#     return args

# def get_time(t):
#     try:
#         t = float(t)
#         return t
#     except:
#         t = str(t).replace('"', '')
#         t = time.mktime(time.strptime(t,'%Y-%m-%d %H:%M:%S'))
#         t = int(t/3600)
#         return t

# def generate_file_for_each_patient(args, features_csv):
#     selected_indices = []
#     initial_dir = args['initial_dir']
#     os.system('rm -r ' + initial_dir)
#     os.mkdir(initial_dir)
#     mkdir(initial_dir)
#     with open(features_csv, 'r') as f:
#       # get length of f
#       file_length = sum(1 for line in f)
#       print(f'There are {file_length} lines')
#       # reset pointer
#       f.seek(0)
#       for i_line, line in enumerate(f):
#         if i_line % 100 == 0:
#           print(f'Processing line {i_line} / {file_length}')
#         if i_line:
#           line_data = line.strip().split(',')
#           assert len(line_data) == len(feat_list)
#           new_line_data = [line_data[i_feat] for i_feat in selected_indices]
#           new_line = ','.join(new_line_data)
#           p_file = os.path.join(initial_dir, line_data[0] + '.csv')
#           if not os.path.exists(p_file):
#             with open(p_file, 'w') as filehandle:
#               filehandle.write(new_head)
#               filehandle.close()
#           filehandle = open(p_file, 'a')
#           filehandle.write('\n' + new_line)
#           filehandle.close()
#         else:
#           feat_list = csv_split(line.strip())
#           feat_list = [f.strip('"') for f in feat_list]
#           print('There are {:d} features.'.format(len(feat_list)))
#           print(feat_list)
#           if len(selected_indices) == 0:
#               selected_indices = range(1, len(feat_list))
#               selected_feat_list = [feat_list[i_feat].replace('"','').replace(',', ';'
#               new_head = ','.join(selected_feat_list)


# def resample_data(args, delta=1, ignore_time=-48):
#     resample_dir = args['resample_dir']
#     initial_dir = args['initial_dir']

#     os.system('rm -r ' + resample_dir)
#     os.mkdir(resample_dir)

#     count_intervals = [0, 0]
#     count_dict = dict()
#     two_sets = [set(), set()]
#     for i_fi, fi in enumerate(tqdm(os.listdir(initial_dir))):
```

```
#         time_line_dict = dict()
#         for i_line, line in enumerate(open(os.path.join(initial_dir, fi))):
#             if i_line:
#                 if len(line.strip()) == 0:
#                     continue
#                 line_data = line.strip().split(',')
#                 assert len(line_data) == len(feat_list)
#                 ctime = get_time(line_data[0])
#                 ctime = delta * int(float(ctime) / delta)
#                 if ctime not in time_line_dict:
#                     time_line_dict[ctime] = []
#                 time_line_dict[ctime].append(line_data)
#             else:
#                 feat_list = line.strip().split(',')
#                 feat_list[0] = 'time'

#         with open(os.path.join(resample_dir, fi), 'w') as wf:
#           wf.write(','.join(feat_list))
#           last_time = None
#           vis = 0
#           max_t = max(time_line_dict)
#           for t in sorted(time_line_dict):
#               if t - max_t < ignore_time:
#                   continue
#               line_list = time_line_dict[t]
#               new_line = line_list[0]
#               for line_data in line_list:
#                   for iv, v in enumerate(line_data):
#                       if len(v.strip()):
#                           new_line[iv] = v
#               new_line[0] = str(t - max_t)
#               new_line = '\n' + ','.join(new_line)
#               wf.write(new_line)

#               if last_time is not None:
#                   delta_t = t - last_time
#                   if delta_t > delta:
#                       vis = 1
#                       count_intervals[0] += 1
#                       count_dict[t - last_time] = count_dict.get(t - last_time, 0) + 1
#                       two_sets[0].add(fi)
#                   two_sets[1].add(fi)
#                   count_intervals[1] += 1
#               last_time = t
#           wf.close()
#     print('There are {:d}/{:d} collections data with intervals > {:d}.'.format(count_i
#     print('There are {:d}/{:d} patients with intervals > {:d}.'.format(len(two_sets[0]


# def generate_feature_dict(args):
#     resample_dir = args['resample_dir']
#     files = sorted(glob(os.path.join(resample_dir, '*')))
#     feature_value_dict = dict()
#     feature_missing_dict = dict()
#     for ifi, fi in enumerate(tqdm(files)):
#         if 'csv' not in fi:
#             continue
#         for iline, line in enumerate(open(fi)):
#             line = line.strip()
#             if iline == 0:
#                 feat_list = line.split(',')
#             else:
#                 data = line.split(',')
#                 for iv, v in enumerate(data):
#                     if v in ['NA', '']:
#                         continue
#                     else:
#                         feat = feat_list[iv]
#                         if feat not in feature_value_dict:
#                             feature_value_dict[feat] = []
#                         feature_value_dict[feat].append(float(v))
#     feature_mm_dict = dict()
#     feature_ms_dict = dict()

#     feature_range_dict = dict()
#     len_time = max([len(v) for v in feature_value_dict.values()])
#     for feat, vs in feature_value_dict.items():
#         vs = sorted(vs)
#         value_split = []
#         for i in range(args['split_num']):
#             n = int(i * len(vs) / args['split_num'])
#             value_split.append(vs[n])
#         value_split.append(vs[-1])
#         feature_range_dict[feat] = value_split


#         n = int(len(vs) / args['split_num'])
#         feature_mm_dict[feat] = [vs[n], vs[-n - 1]]
#         feature_ms_dict[feat] = [np.mean(vs), np.std(vs)]

#         feature_missing_dict[feat] = 1.0 - 1.0 * len(vs) / len_time

#     json.dump(feature_mm_dict, open(os.path.join(args['files_dir'], 'feature_mm_dict.j
#     json.dump(feature_ms_dict, open(os.path.join(args['files_dir'], 'feature_ms_dict.j
#     json.dump(feat_list, open(os.path.join(args['files_dir'], 'feature_list.json'), 'w
#     json.dump(feature_missing_dict, open(os.path.join(args['files_dir'], 'feature_miss
#     json.dump(feature_range_dict, open(os.path.join(args['files_dir'], 'feature_value_


# def split_data_to_ten_set(args):
#     resample_dir = args['resample_dir']
#     files = sorted(glob(os.path.join(resample_dir, '*')))
#     np.random.shuffle(files)
#     splits = []
#     for i in range(10):
#         st = int(len(files) * i / 10)
#         en = int(len(files) * (i+1) / 10)
#         splits.append(files[st:en])
#     json.dump(splits, open(os.path.join(args['files_dir'], 'splits.json'), 'w'))


# def generate_label_dict(args, task):
#     label_dict = dict()
#     for i_line, line in enumerate(open(os.path.join(args['data_dir'], '%s.csv' % task)
#         if i_line:
#             data = line.strip().split(',')
#             pid = data[0]
#             label = ''.join(data[1:])
#             pid = str(int(float(pid)))
#             label_dict[pid] = label
#     with open(os.path.join(args['files_dir'], '%s_dict.json' % task), 'w') as json_fil
#       json.dump(label_dict, json_file)



# def generate_demo_dict(args, demo_csv):
#     demo_dict = dict()
#     demo_index_dict = dict()
#     for i_line, line in enumerate(open(demo_csv)):
#         if i_line:
#             data = line.strip().split(',')
#             pid = str(int(float(data[0])))
#               demo_dict[pid] = []
#               for demo in data[1:]:
#                   if demo not in demo_index_dict:
#                       demo_index_dict[demo] = len(demo_index_dict)
#                   demo_dict[pid].append(demo_index_dict[demo])
#       with open(os.path.join(args['files_dir'], 'demo_dict.json'), 'w') as json_file:
#         json.dump(demo_dict, json_file)
#       with open(os.path.join(args['files_dir'], 'demo_index_dict.json'), 'w') as json_fi
#         json.dump(demo_index_dict, json_file)


# def main():
#     args = parse_args()
#     args['files_dir'] = os.path.join(args['data_dir'], 'files')
#     args['initial_dir'] = os.path.join(args['data_dir'], 'initial_data')
#     args['resample_dir'] = os.path.join(args['data_dir'], 'resample_dir')
#     args['split_num'] = 4000
#     print(args.items())

#     for x in ['files', 'initial_data', 'resample_dir']:
#       if x not in os.listdir(args['data_dir']):
#         if x == 'files':
#           os.mkdir(args['files_dir'])
#         elif x == 'initial_data':
#           os.mkdir(args['initial_dir'])
#         elif x == 'resample_dir':
#           os.mkdir(args['resample_dir'])

#     features_csv = os.path.join(args['data_dir'], 'features.csv')
#     demo_csv = os.path.join(args['data_dir'], 'demo.csv')
#     for task in ['mortality', 'readmit', 'llos']:
#         generate_label_dict(args, task)
#     generate_demo_dict(args, demo_csv)
#     generate_file_for_each_patient(args, features_csv)
#     resample_data(args)
#     generate_feature_dict(args)
#     split_data_to_ten_set(args)

# main()
```

Preprocessing Step 7: Docs2Vec

In [21]:
```
import pandas as pd
import numpy as np
from tqdm import tqdm
from sklearn.utils import shuffle
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
import json
import argparse

def parse_args(epochs=None, phase=None):
    args = {}
    if epochs is None:
      args['epochs'] = 30
    if phase is None:
      args['phase'] = 'infer'
    return args


processed_root = "/content/drive/MyDrive/mimic-iii_processed_data"
files_root = "/content/drive/MyDrive/mimic-iii_processed_data/files"
models_root = "/content/drive/MyDrive/models"
args = parse_args()
```

```
#df = pd.read_csv(f'{processed_root}/earlynotes_demo.csv')
#df['text'] = df['text'].astype(str).apply(text2words)
#print(os.listdir(models_root))
```

DATA PIPELINE

In [22]:
```
#vector_dict = json.load(open(f'{args.data_dir}/files/vector_dict.json', 'r'))

def find_index(v, vs, i=0, j=-1):
    if j == -1:
        j = len(vs) - 1

    if v > vs[j]:
        return j + 1
    elif v < vs[i]:
        return i
    elif j - i == 1:
        return j

    k = int((i + j)/2)
    if v <= vs[k]:
        return find_index(v, vs, i, k)
    else:
        return find_index(v, vs, k, j)


class DataBowl(Dataset):
    def __init__(self, args, files, phase='train'):
        assert (phase == 'train' or phase == 'valid' or phase == 'test')
        self.args = args
        self.phase = phase
        self.files = files
        self.feature_mm_dict = json.load(
            open(os.path.join(args.files_dir, 'feature_mm_dict.json'), 'r'))
        self.feature_value_dict = json.load(open(os.path.join(
            args.files_dir, 'feature_value_dict_%d.json' % args.split_num), 'r'))
        self.demo_dict = json.load(
            open(os.path.join(args.files_dir, 'demo_dict.json'), 'r'))
        self.label_dict = json.load(
            open(os.path.join(args.files_dir, '%s_dict.json' % args.task), 'r'))

        print('Use the last %d collections data' % args.n_visit)

    def map_input(self, value, feat_list, feat_index):
        index_start = (feat_index + 1) * (1 + self.args.split_num) + 1

        if value in ['NA', '']:
            return 0
        else:
            value = float(value)
            vs = self.feature_value_dict[feat_list[feat_index]][1:-1]
            v = find_index(value, vs) + index_start
            return v

    def map_output(self, value, feat_list, feat_index):
        if value in ['NA', '']:
            return 0
        else:
            value = float(value)
            minv, maxv = self.feature_mm_dict[feat_list[feat_index]]
            if maxv <= minv:
                print(feat_list[feat_index], minv, maxv)
            assert maxv > minv
            v = (value - minv) / (maxv - minv)
            v = max(0, min(v, 1))
            return v

    def get_mm_item(self, idx):
        input_file = self.files[idx]
        print(input_file)
        pid = input_file.split('/')[-1].split('.')[0]

        if input_file in args.resample_dir:
          with open(input_file) as f:
              input_data = f.read().strip().split('\n')
        else:
          input_data = []

        time_list, input_list = [], []

        for iline in range(len(input_data)):
            inp = input_data[iline].strip()
            if iline == 0:
                feat_list = inp.split(',')
            else:
                in_vs = inp.split(',')
                ctime = int(inp.split(',')[0])
                input = []
                for i, iv in enumerate(in_vs):
                    if self.args.use_ve:
                        input.append(self.map_input(iv, feat_list, i))
                    else:
                        input.append(self.map_output(iv, feat_list, i))
                input_list.append(input)
                time_list.append(- int(ctime))

        if len(input_list) < self.args.n_visit:
            for _ in range(self.args.n_visit - len(input_list)):
                # pad empty visit
                vs = [0 for _ in range(self.args.input_size + 1)]
                input_list = [vs] + input_list
                time_list = [time_list[0]] + time_list
        else:
            if self.use_first_records:
                input_list = input_list[: self.args.n_visit]
                time_list = time_list[: self.args.n_visit]
            else:
                input_list = input_list[-self.args.n_visit:]
                time_list = time_list[-self.args.n_visit:]

        if self.args.value_embedding == 'no' or self.args.use_ve == 0:
            input_list = np.array(input_list, dtype=np.float32)
        else:
            input_list = np.array(input_list, dtype=np.int64)
        time_list = np.array(time_list, dtype=np.int64) + 1
        assert time_list.min() >= 0
        if self.args.value_embedding != 'no':
            input_list = input_list[:, 1:]
        else:
            input_list = input_list.transpose()

        label = np.array([int(l)
                          for l in self.label_dict[pid]], dtype=np.float32)
        # demo = np.array([self.demo_dict[pid] for _ in range(self.args.n_visit)], dtype
        demo = np.array(self.demo_dict.get(pid, 0), dtype=np.int64)

        # content = self.unstructure_dict.get(pid, [])
        # while len(content) < self.max_length:
        #     content.append(0)
        # content = content[: self.max_length]
        # content = np.array(content, dtype=np.int64)
        content = vector_dict[pid]
        while len(content) < 12:
            content.append([0] * 200)
        content = content[:12]
        content = np.array(content, dtype=np.float32)
        # content = np.mean(content, axis=0)

        return torch.from_numpy(input_list), torch.from_numpy(time_list), torch.from_num

    def __getitem__(self, idx):
        return self.get_mm_item(idx)

    def __len__(self):
        return len(self.files)
```

METHODOLOGY (MODEL)

Model Description

The research introduces a multi-modal neural network architecture designed to enhance predictive modeling by integrating both structured and unstructured data from Electronic Health Records (EHRs). The model capitalizes on the strengths of convolutional neural networks (CNNs) and long short-term memory networks (LSTMs) to process and learn from the temporal dynamics and textual complexity found within clinical health datasets.

1. **Components**:

   - **Static Information Encoder**: Encodes static categorical features such as patient demos and admission-related information by way of one-hot encoding.
   - **Temporal Signals**: Utilize a sequential layer (CNN and/or LSTM) to model time-series clinical data, extracting vital patterns and signals from medical measurements.
   - **Sequential Notes Representation**: Use of document embeddings combined with sequential neural network architecture to handle long sequences found in clinical notes, enhancing the ability to capture relevant medical contexts and details.

2. **Data Fusion**:

   - The model integrates the processed data streams into a unified patient representation, which combines the encoded static information, temporal features, and finally the text-based insights.

LSTM

In [23]:
```
def value_embedding_data(d = 200, split = 200):
    vec = np.array([np.arange(split) * i for i in range(int(d/2))], dtype=np.float32).tr
    vec = vec / vec.max()
    embedding = np.concatenate((np.sin(vec), np.cos(vec)), 1)
    embedding[0, :d] = 0
    embedding = torch.from_numpy(embedding)
    return embedding


class LSTM(nn.Module):
    def __init__(self, args):
        super(LSTM, self).__init__()
        self.args = args

        # unstructure
        if args.use_unstructure:
            self.vocab_embedding = nn.Embedding (args.unstructure_size, args.embed_size
            self.vocab_lstm = nn.LSTM ( input_size=args.embed_size,
                                  # hidden_size=args.hidden_size,
                                  hidden_size=1,
                                  num_layers=args.num_layers,
                                  batch_first=True,
                                  bidirectional=True)
            self.vocab_mapping = nn.Sequential(
                    nn.Linear(args.embed_size * 2, args.embed_size),
                    nn.ReLU ( ),
                    nn.Dropout ( 0.1),
                    nn.Linear(args.embed_size, args.embed_size),
                    )
            self.cat_output = nn.Sequential (
                    nn.Linear (args.rnn_size * 3, args.rnn_size),
                    nn.ReLU ( ),
                    nn.Dropout ( 0.1),
                    nn.Linear ( args.rnn_size, output_size),
                    )
            self.cat_output = nn.Sequential (
                    nn.ReLU ( ),
                    nn.Dropout ( 0.1),
                    nn.Linear (args.rnn_size * 3, output_size),
                    )

        if args.value_embedding == 'no':
            self.embedding = nn.Linear(args.input_size, args.embed_size)
        else:
            self.embedding = nn.Embedding (args.vocab_size, args.embed_size )
        self.lstm1 = nn.LSTM (input_size=args.embed_size,
                              hidden_size=args.hidden_size,
                              num_layers=args.num_layers,
                              batch_first=True,
                              bidirectional=True)
        self.lstm2 = nn.LSTM (input_size=args.embed_size,
                              hidden_size=args.hidden_size,
                              num_layers=args.num_layers,
                              batch_first=True,
                              bidirectional=True)
        self.dd_embedding = nn.Embedding (args.n_ehr, args.embed_size )
        self.value_embedding = nn.Embedding.from_pretrained(value_embedding_data(args.em
        self.value_mapping = nn.Sequential(
                nn.Linear ( args.embed_size * 2, args.embed_size),
                nn.ReLU ( ),
                nn.Dropout ( 0.1),
                )
        self.dd_mapping = nn.Sequential(
                nn.Linear ( args.embed_size, args.embed_size),
                nn.ReLU ( ),
                nn.Dropout(0.1),
                nn.Linear ( args.embed_size, args.embed_size),
                nn.ReLU ( ),
                nn.Dropout(0.1),
                )
        self.dx_mapping = nn.Sequential(
                nn.Linear ( args.embed_size * 2, args.embed_size),
                nn.ReLU ( ),
                nn.Linear ( args.embed_size, args.embed_size),
                nn.ReLU ( ),
                )

        self.tv_mapping = nn.Sequential (
            nn.Linear ( args.embed_size * 2, args.embed_size),
            nn.ReLU ( ),
            nn.Linear ( args.embed_size, args.embed_size),
            nn.ReLU ( ),
            nn.Dropout ( 0.1),
        )
        self.relu = nn.ReLU ( )

        lstm_size = args.rnn_size

        lstm_size *= 2
        self.output_mapping = nn.Sequential (
            nn.Linear (lstm_size, args.rnn_size),
            nn.ReLU ( ),
            nn.Linear (args.rnn_size, args.rnn_size),
            nn.ReLU ( )
        )

        self.output = nn.Sequential (
            nn.Linear (args.rnn_size * 2, args.rnn_size),
            nn.ReLU ( ),
            nn.Dropout ( 0.1),
            nn.Linear ( args.rnn_size, output_size),
        )
        self.pooling = nn.AdaptiveMaxPool1d(1)

        self.one_output = nn.Sequential (
                # nn.Linear (args.embed_size * 3, args.embed_size),
                # nn.ReLU ( ),
                nn.Dropout ( 0.1),
                nn.Linear ( args.embed_size, output_size),
            )


    def visit_pooling(self, x):
        output = x
        size = output.size()
        output = output.view(size[0] * size[1], size[2], output.size(3))    # (64*30, 13
        output = torch.transpose(output, 1,2).contiguous()                  # (64*30, 20
        output = self.pooling(output)                                       # (64*30, 20
        output = output.view(size[0], size[1], size[3])                     # (64, 30, 2
        return output

    def value_order_embedding(self, x):
        size = list(x[0].size())               # (64, 30, 13)
        index, value = x
        xi = self.embedding(index.view(-1))          # (64*30*13, 200)
        # xi = xi * (value.view(-1).float() + 1.0 / self.args.split_num)
        xv = self.value_embedding(value.view(-1))    # (64*30*13, 200)
        x = torch.cat((xi, xv), 1)                   # (64*30*13, 1024)
        x = self.value_mapping(x)                    # (64*30*13, 200)
        size.append(-1)
        x = x.view(size)                       # (64, 30, 13, 200)
        return x


    def forward(self, x, t, dd, content=None):

        if 0 and content is not None:
            content, _ = self.lstm1(content)
            content = self.vocab_mapping(content)
            content = torch.transpose(content, 1, 2).contiguous()
            content = self.pooling(content)
            content = content.view((content.size(0), -1))
            return self.one_output(content)
        # value embedding
        x = self.value_order_embedding(x)
        x = self.visit_pooling(x)

        # demo embedding
        dsize = list(dd.size()) + [-1]
        d = self.dd_embedding(dd.view(-1)).view(dsize)
        d = self.dd_mapping(d)
        d = torch.transpose(d, 1,2).contiguous()                  # (64*30, 200, 100)
        d = self.pooling(d)
        d = d.view((d.size(0), -1))

        # x = torch.cat((x, d), 2)
        # x = self.dx_mapping(x)

        # time embedding
        # t = self.value_embedding(t)
        # x = self.tv_mapping(torch.cat((x, t), 2))

        # lstm
        lstm_out, _ = self.lstm2( x )             # (64, 30, 1024)
        output = self.output_mapping(lstm_out)
        output = torch.transpose(output, 1,2).contiguous()                  # (64*30, 20
        # print('ouput.size', output.size())
        output = self.pooling(output)                                       # (64*30, 20
        output = output.view((output.size(0), -1))
        out = self.output(torch.cat((output, d), 1))

        # unstructure
        if content is not None:
            # print(content.size())   # [64, 1000]
            content, _ = self.lstm1(content)
            content = self.vocab_mapping(content)
            content = torch.transpose(content, 1, 2).contiguous()
            content = self.pooling(content)
            content = content.view((content.size(0), -1))
            out = self.cat_output(torch.cat((output, content, d), 1))


        return out
```

CNN

In [24]:
```
import os
import json
import torch
from torch import nn
import torch.nn.functional as F
from torch.autograd import *
import numpy as np
import sys
output_size = 1

def value_embedding_data(d = 200, split = 200):
    vec = np.array([np.arange(split) * i for i in range(int(d/2))], dtype=np.float32).tr
    vec = vec / vec.max()
    embedding = np.concatenate((np.sin(vec), np.cos(vec)), 1)
    embedding[0, :d] = 0
    embedding = torch.from_numpy(embedding)
    return embedding


def conv3(in_channels, out_channels, stride=1, kernel_size=3):
```

```python
    return nn.Conv1d(in_channels, out_channels, kernel_size=kernel_size,
                     stride=stride, padding=1, bias=False)

class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(ResidualBlock, self).__init__()
        self.conv1 = conv3(in_channels, out_channels, stride)
        self.bn1 = nn.BatchNorm1d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3(out_channels, out_channels)
        self.bn2 = nn.BatchNorm1d(out_channels)
        self.downsample = downsample

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out

class CNN(nn.Module):
    def __init__(self, args):
        super(CNN, self).__init__()
        self.args = args


        if args.value_embedding == 'no':
            self.embedding = nn.Linear(args.input_size, args.embed_size)
        else:
            self.embedding = nn.Embedding (args.vocab_size, args.embed_size )
        self.dd_embedding = nn.Embedding (args.n_ehr, args.embed_size )
        self.value_embedding = nn.Embedding.from_pretrained(value_embedding_data(args.em
        self.value_mapping = nn.Sequential(
                nn.Linear ( args.embed_size * 2, args.embed_size),
                nn.ReLU ( ),
                nn.Dropout ( 0.1),
                )
        self.dd_mapping = nn.Sequential(
                nn.Linear ( args.embed_size, args.embed_size),
                nn.ReLU ( ),
                nn.Dropout(0.1),
                nn.Linear ( args.embed_size, args.embed_size),
                nn.ReLU ( ),
                nn.Dropout(0.1),
                )
        self.dx_mapping = nn.Sequential(
                nn.Linear ( args.embed_size * 2, args.embed_size),
                nn.ReLU ( ),
                nn.Linear ( args.embed_size, args.embed_size),
                nn.ReLU ( ),
                )

        self.tv_mapping = nn.Sequential (
            nn.Linear ( args.embed_size * 2, args.embed_size),
            nn.ReLU ( ),
            nn.Linear ( args.embed_size, args.embed_size),
            nn.ReLU ( ),
            nn.Dropout ( 0.1),
        )
        self.relu = nn.ReLU ( )
        lstm_size = args.rnn_size

        lstm_size *= 2
        self.output_mapping = nn.Sequential (
            nn.Linear (lstm_size, args.rnn_size),
            nn.ReLU ( ),
            nn.Linear (args.rnn_size, args.rnn_size),
            nn.ReLU ( )
        )

        self.cat_output = nn.Sequential (
            nn.Linear (args.rnn_size * 2, args.rnn_size),
            nn.ReLU ( ),
            nn.Dropout ( 0.1),
            nn.Linear ( args.rnn_size, output_size),
        )
        self.output = nn.Sequential (
            nn.Linear (args.rnn_size, args.rnn_size),
            nn.ReLU ( ),
            nn.Dropout ( 0.1),
            nn.Linear ( args.rnn_size, output_size),
        )
        self.pooling = nn.AdaptiveMaxPool1d(1)


        layers = [1, 2, 2]
        embed_size = args.embed_size
        block = ResidualBlock
        self.in_channels = embed_size
        self.bn1 = nn.BatchNorm1d(embed_size)
        self.bn2 = nn.BatchNorm1d(embed_size)
        self.bn3 = nn.BatchNorm1d(embed_size)
        self.relu = nn.ReLU(inplace=True)
        self.layer1 = self.make_layer(block, embed_size, layers[0], 2)
        self.layer2 = self.make_layer(block, embed_size, layers[1], 2)
        self.layer3 = self.make_layer(block, embed_size, layers[2], 2)



        # unstructure
        if args.use_unstructure:
            self.vocab_embedding = nn.Embedding (args.unstructure_size+10, args.embed_si
            # self.vocab_layer = self.make_layer(block, embed_size, layers[0], 2)
            self.vocab_layer = nn.Sequential(
                    nn.Dropout(0.2),
                    conv3(embed_size, embed_size, 2, 2),
                    nn.BatchNorm1d(embed_size),
                    nn.Dropout(0.2),
                    nn.ReLU(),
                    # conv3(embed_size, embed_size, 2, 3),
                    # nn.BatchNorm1d(embed_size),
                    # nn.Dropout(0.1),
                    # nn.ReLU(),
                    # conv3(embed_size, embed_size, 2, 3),
                    # nn.BatchNorm1d(embed_size),
                    # nn.Dropout(0.1),
                    # nn.ReLU(),
                    )
            self.vocab_output = nn.Sequential (
                nn.ReLU ( ),
                nn.Dropout ( 0.1),
                nn.Linear (args.embed_size * 3, 3 * args.embed_size),
                nn.ReLU ( ),
                nn.Dropout ( 0.1),
                nn.Linear ( 3 * args.embed_size, output_size),
            )
            self.one_output = nn.Sequential (
                # nn.Linear (args.embed_size * 3, args.embed_size),
                # nn.ReLU ( ),
                nn.Dropout ( 0.1),
                nn.Linear ( args.embed_size, output_size),
            )


    def visit_pooling(self, x):
        output = x
        size = output.size()
        output = output.view(size[0] * size[1], size[2], output.size(3))    # (64*30, 13
        output = torch.transpose(output, 1,2).contiguous()                  # (64*30, 20
        output = self.pooling(output)                                       # (64*30, 20
        output = output.view(size[0], size[1], size[3])                     # (64, 30, 2
        return output

    def value_order_embedding(self, x):
        size = list(x[0].size())              # (64, 30, 13)
        index, value = x
        xi = self.embedding(index.view(-1))          # (64*30*13, 200)
        # xi = xi * (value.view(-1).float() + 1.0 / self.args.split_num)
        xv = self.value_embedding(value.view(-1))    # (64*30*13, 200)
        x = torch.cat((xi, xv), 1)                   # (64*30*13, 1024)
        x = self.value_mapping(x)                    # (64*30*13, 200)
        size.append(-1)
        x = x.view(size)                    # (64, 30, 13, 200)
        return x


    def make_layer(self, block, out_channels, blocks, stride=1):
        downsample = None
        if (stride != 1) or (self.in_channels != out_channels):
            downsample = nn.Sequential(
                conv3(self.in_channels, out_channels, stride=stride),
                nn.BatchNorm1d(out_channels))
        layers = []
        layers.append(block(self.in_channels, out_channels, stride, downsample))
        self.in_channels = out_channels
        for i in range(1, blocks):
            layers.append(block(out_channels, out_channels))
        return nn.Sequential(*layers)



    def forward(self, x, t, dd, content=None):

        if content is not None:
            # content = self.vocab_embedding(content).transpose(1,2)
            content = self.vocab_layer(content.transpose(1,2))
            content = self.pooling(content)                                     # (64*
            content = content.view((content.size(0), -1))
            return self.one_output(content)

        # value embedding
        x = self.value_order_embedding(x)
        x = self.visit_pooling(x)


        # time embedding
        # t = self.value_embedding(t)
        # x = self.tv_mapping(torch.cat((x, t), 2))

        # cnn
        x = torch.transpose(x, 1, 2,).contiguous()
```

```
        out = self.bn1(x)
        out = self.relu(out)
        out = self.layer1(out)
        # out = self.layer2(out)
        # out = self.layer3(out)

        output = self.pooling(out)                                       # (64*30, 200,
        output = output.view((output.size(0), -1))


        if len(dd.size()) > 1:
            # demo embedding
            dsize = list(dd.size()) + [-1]
            d = self.dd_embedding(dd.view(-1)).view(dsize)
            d = self.dd_mapping(d)
            d = torch.transpose(d, 1,2).contiguous()                   # (64*30, 200, 100
            d = self.pooling(d)
            d = d.view((d.size(0), -1))
            output = torch.cat((output, d), 1)
            out = self.cat_output(output)
        # else:
        #     out = self.output(output)

        if content is not None:
            # content = self.vocab_embedding(content)
            content = self.vocab_layer(content.transpose(1,2))
            content = self.pooling(content)                                        # (64*
            content = content.view((content.size(0), -1))
            # content = self.one_output(content) + 0.3 * out
            # out = content + out
            # out = content
            output = torch.cat((output, content), 1)
            out = self.vocab_output(output)

        return out
```

CRITERION DEFINTION (& OTHER RELATIVE METRICS)

In [25]:
```
def hard_mining(neg_output, neg_labels, num_hard, largest=True):
    num_hard = min(max(num_hard, 10), len(neg_output))
    _, idcs = torch.topk(neg_output, min(num_hard, len(neg_output)), largest=largest)
    neg_output = torch.index_select(neg_output, 0, idcs)
    neg_labels = torch.index_select(neg_labels, 0, idcs)
    return neg_output, neg_labels


class Loss(nn.Module):
    def __init__(self, hard_mining):
        super(Loss, self).__init__()
        self.classify_loss = nn.BCELoss()
        self.hard_mining = hard_mining
        self.sigmoid = nn.Sigmoid()

    def forward(self, prob, labels, train=True):

        prob = self.sigmoid(prob)

        pos_ind = labels > 0.5
        neg_ind = labels < 0.5
        pos_label = labels[pos_ind]
        neg_label = labels[neg_ind]
        pos_prob = prob[pos_ind]
        neg_prob = prob[neg_ind]
        pos_loss, neg_loss = 0, 0

        # hard mining
        num_hard_pos = 10
        num_hard_neg = 18
        if self.hard_mining:
            pos_prob, pos_label= hard_mining(pos_prob, pos_label, num_hard_pos, largest=
            neg_prob, neg_label= hard_mining(neg_prob, neg_label, num_hard_neg, largest=

        if len(pos_prob):
            pos_loss = 0.5 * self.classify_loss(pos_prob, pos_label)

        if len(neg_prob):
            neg_loss = 0.5 * self.classify_loss(neg_prob, neg_label)
        classify_loss = pos_loss + neg_loss

        prob = prob.data.cpu().numpy() > 0.5
        labels = labels.data.cpu().numpy()
        pos_l = (labels==1).sum()
        neg_l = (labels==0).sum()
        pos_p = (prob + labels == 2).sum()
        neg_p = (prob + labels == 0).sum()

        return [classify_loss, pos_p, pos_l, neg_p, neg_l]


class MultiClassLoss(nn.Module):
    def __init__(self, hard_mining):
        super(MultiClassLoss, self).__init__()
        self.classify_loss = nn.BCELoss()
        self.hard_mining = hard_mining
        self.sigmoid = nn.Sigmoid()


    def forward(self, prob, labels, train=True):
        prob = self.sigmoid(prob)
        classify_loss, pos_p, pos_l, neg_p, neg_l = 0, 0, 0, 0, 0

        prob_list = prob
        labels_list = labels
        for i in range(prob.size(1)):
            prob = prob_list[:, i]
            labels = labels_list[:, i]

            pos_ind = labels > 0.5
            neg_ind = labels < 0.5
            pos_label = labels[pos_ind]
            neg_label = labels[neg_ind]
            pos_prob = prob[pos_ind]
            neg_prob = prob[neg_ind]
            pos_loss, neg_loss = 0, 0

            # hard mining
            num_hard_pos = 10
            num_hard_neg = 18
            if self.hard_mining:
                pos_prob, pos_label= hard_mining(pos_prob, pos_label, num_hard_pos, larg
                neg_prob, neg_label= hard_mining(neg_prob, neg_label, num_hard_neg, larg

            if len(pos_prob):
                pos_loss = 0.5 * self.classify_loss(pos_prob, pos_label)

            if len(neg_prob):
                neg_loss = 0.5 * self.classify_loss(neg_prob, neg_label)
            classify_loss = classify_loss + pos_loss + neg_loss

            # stati number
            prob = prob.data.cpu().numpy() > 0.5
            labels = labels.data.cpu().numpy()
            pos_l += (labels==1).sum()
            neg_l += (labels==0).sum()
            pos_p += (prob + labels == 2).sum()
            neg_p += (prob + labels == 0).sum()

        return [classify_loss, pos_p, pos_l, neg_p, neg_l]
```

In [26]:
```
import numpy as np
import os
import torch
from sklearn import metrics


def compute_nRMSE(pred, label, mask):
    '''
    same as 3dmice
    '''
    assert pred.shape == label.shape == mask.shape

    missing_indices = mask==1
    missing_pred = pred[missing_indices]
    missing_label = label[missing_indices]
    missing_rmse = np.sqrt(((missing_pred - missing_label) ** 2).mean())

    init_indices = mask==0
    init_pred = pred[init_indices]
    init_label = label[init_indices]
    init_rmse = np.sqrt(((init_pred - init_label) ** 2).mean())

    metric_list = [missing_rmse, init_rmse]
    for i in range(pred.shape[2]):
        apred = pred[:,:,i]
        alabel = label[:,:, i]
        amask = mask[:,:, i]

        mrmse, irmse = [], []
        for ip in range(len(apred)):
            ipred = apred[ip]
            ilabel = alabel[ip]
            imask = amask[ip]

            x = ilabel[imask>=0]
            if len(x) == 0:
                continue

            minv = ilabel[imask>=0].min()
            maxv = ilabel[imask>=0].max()
            if maxv == minv:
                continue

            init_indices = imask==0
            init_pred = ipred[init_indices]
            init_label = ilabel[init_indices]

            missing_indices = imask==1
            missing_pred = ipred[missing_indices]
            missing_label = ilabel[missing_indices]

            assert len(init_label) + len(missing_label) >= 2
            if len(init_pred) > 0:
                init_rmse = np.sqrt((((init_pred - init_label) / (maxv - minv)) ** 2).me
                irmse.append(init_rmse)

            if len(missing_pred) > 0:
                missing_rmse = np.sqrt((((missing_pred - missing_label)/ (maxv - minv))
                mrmse.append(missing_rmse)

        metric_list.append(np.mean(mrmse))
        metric_list.append(np.mean(irmse))

    metric_list = np.array(metric_list)


    metric_list[0] = np.mean(metric_list[2:][::2])
    metric_list[1] = np.mean(metric_list[3:][::2])

    return metric_list


def save_model(p_dict):
    args = p_dict['args']
    model = p_dict['model']
    state_dict = model.state_dict()
    for key in state_dict.keys():
        state_dict[key] = state_dict[key].cpu()
    all_dict = {
            'epoch': p_dict['epoch'],
            'args': p_dict['args'],
            'best_metric': p_dict['best_metric'],
            'state_dict': state_dict
            }
    torch.save(all_dict, args.model_path)

def load_model(p_dict, model_file):
    all_dict = torch.load(model_file)
    p_dict['epoch'] = all_dict['epoch']
    # p_dict['args'] = all_dict['args']
    p_dict['best_metric'] = all_dict['best_metric']
    # for k,v in all_dict['state_dict'].items():
    #     p_dict['model_dict'][k].load_state_dict(all_dict['state_dict'][k])
    p_dict['model'].load_state_dict(all_dict['state_dict'])

def compute_auc(labels, probs):
    fpr, tpr, thr = metrics.roc_curve(labels, probs)
    return metrics.auc(fpr, tpr)

def compute_metric(labels, probs):
    labels = np.array(labels)
    probs = np.array(probs)
    fpr, tpr, thresholds = metrics.roc_curve(labels, probs)
    auc = metrics.auc(fpr, tpr)
    aupr = metrics.average_precision_score(labels, probs)
    optimal_threshold = thresholds[np.argmax(tpr - fpr)]
    preds = [1 if prob >= optimal_threshold else 0 for prob in probs]
    tn, fp, fn, tp = metrics.confusion_matrix(labels, preds).ravel()
    precision = 1.0 * (tp / (tp + fp))
    sen = 1.0 * (tp / (tp + fn))  # recall
    spec = 1.0 * (tn / (tn + fp))
    f1 = metrics.f1_score(labels, preds)
    return precision, sen, spec, f1, auc, aupr
```

In [27]:
```
def cal_metric(y_true, probs):
    fpr, tpr, thresholds = metrics.roc_curve(y_true, probs)
    optimal_idx = np.argmax(np.sqrt(tpr * (1-fpr)))
    optimal_threshold = thresholds[optimal_idx]
    preds = (probs > optimal_threshold).astype(int)
    auc = metrics.roc_auc_score(y_true, probs)
    auprc = metrics.average_precision_score(y_true, probs)
    f1 = metrics.f1_score(y_true, preds)
    return f1, auc, auprc
```

METHODOLOGY (TRAINING)

Argument Parsing

In [28]:
```
parser = argparse.ArgumentParser(description='clinical fusion help')
parser.add_argument(
        '--data-dir',
        type=str,
        default='/content/drive/MyDrive/mimic-iii_processed_data/',
        help='selected and preprocessed data directory'
        )

# problem setting
parser.add_argument('--task',
        default='mortality',
        type=str,
        metavar='S',
        help='start from checkpoints')
parser.add_argument(
        '--last-time',
        metavar='last event time',
        type=int,
        default=-4,
        help='last time'
        )
parser.add_argument(
        '--time-range',
        default=10000,
        type=int)
parser.add_argument(
        '--n-code',
        default=8,
        type=int,
        help='at most n codes for same visit')
parser.add_argument(
        '--n-visit',
        default=24,
        type=int,
        help='at most input n visits')



# method seetings
parser.add_argument(
        '--model',
        '-m',
        type=str,
        default='lstm',
        help='model'
        )
parser.add_argument(
        '--split-num',
        metavar='split num',
        type=int,
        default=4000,
        help='split num'
        )
parser.add_argument(
        '--split-nor',
        metavar='split normal range',
        type=int,
        default=200,
        help='split num'
        )
parser.add_argument(
        '--use-glp',
        metavar='use global pooling operation',
        type=int,
        default=0,
        help='use global pooling operation'
        )
parser.add_argument(
        '--use-value',
        metavar='use value embedding as input',
        type=int,
        default=1,
        help='use value embedding as input'
        )
parser.add_argument(
        '--use-cat',
        metavar='use cat for time and value embedding',
        type=int,
        default=1,
        help='use cat or add'
        )


# model parameters
parser.add_argument(
        '--embed-size',
        metavar='EMBED SIZE',
        type=int,
        default=512,
        help='embed size'
        )
parser.add_argument(
        '--rnn-size',
        metavar='rnn SIZE',
        type=int,
        help='rnn size'
        )
parser.add_argument(
        '--hidden-size',
        metavar='hidden SIZE',
        type=int,
        help='hidden size'
        )
parser.add_argument(
        '--num-layers',
        metavar='num layers',
        type=int,
        default=2,
        help='num layers'
        )



# traing process setting
parser.add_argument('--phase',
        default='train',
        type=str,
        help='train/test phase')
parser.add_argument(
        '--batch-size',
        '-b',
        metavar='BATCH SIZE',
        type=int,
        default=64,
        help='batch size'
        )
parser.add_argument('--model-path', type=str, default='models/best.ckpt', help='model pa
parser.add_argument('--resume',
        default='',
        type=str,
        metavar='S',
        help='start from checkpoints')
parser.add_argument(
        '--workers',
        default=8,
        type=int,
        metavar='N',
        help='number of data loading workers (default: 32)')
parser.add_argument('--lr',
        '--learning-rate',
        default=0.0001,
        type=float,
        metavar='LR',
        help='initial learning rate')
parser.add_argument('--epochs',
        default=50,
        type=int,
        metavar='N',
        help='number of total epochs to run')


args = parser.parse_args(args=[])
#args.data_dir = os.path.join(args.data_dir, 'processed')
args.files_dir = os.path.join(args.data_dir, 'files')
args.resample_dir = os.path.join(args.data_dir, 'resample_dir')
args.initial_dir = os.path.join(args.data_dir, 'initial_data')
args.embed_size = 200
args.hidden_size = args.rnn_size = args.embed_size
if torch.cuda.is_available():
    args.gpu = 1
else:
    args.gpu = 0
args.use_ve = 1
args.n_visit = 24
args.use_unstructure = 0
args.value_embedding = 'use_order'
# args.value_embedding = 'no'
print ('epochs,', args.epochs)

args.task = 'mortality'
args.files_dir = args.files_dir
args.data_dir = args.data_dir
```

```
epochs, 50
```

In [29]:

```
demo = True
if not demo:
  epochs = args['epochs']
  splits = range(10)
  data = json.load(open(f'{files_root}/splits.json'))
  train_ids = np.hstack([data[t] for t in splits[:7]])
  train_ids = [x.split('/')[-1] for x in train_ids]
  train_ids = [int(x.split('.')[0]) for x in train_ids]
  train = df[df['hadm_id'].isin(train_ids)]['text'].tolist()

  train_tagged = []
  for idx, text in enumerate(train):
      train_tagged.append(TaggedDocument(text, tags=[str(idx)]))

  model = Doc2Vec(dm=0, vector_size=200, negative=5, alpha=0.025, hs=0, min_count=5, sam
  model.build_vocab([x for x in train_tagged])
  for epoch in tqdm(range(epochs)):
      model.train(shuffle([x for x in train_tagged]), total_examples=len(train_tagged),
      model.alpha -= 0.0002
      model.min_alpha = model.alpha

  model.save(f'{models_root}/doc2vec.model')
```

METHODOLOGY (EVALUATION)

We use the patient representation described in the prior section to predict critical clinical outcomes such as in-hospital mortality, 30-day hospital readmission, and extended hospital stays. The effectiveness of the models will demonstrated through evaluations on the MIMIC-III dataset, where they're *claimed* to outperform traditional approaches that rely solely on structured or unstructured data.

In [30]:
```
# prompt: evaluate trained saved model
try:
  model = Doc2Vec.load(f'{models_root}/doc2vec.model')
  test_ids = np.hstack([data[t] for t in splits[7:]])
  test_ids = [x.split('/')[-1] for x in test_ids]
  test_ids = [int(x.split('.')[0]) for x in test_ids]
  test = df[df['hadm_id'].isin(test_ids)]['text'].tolist()
  test_tagged = []
  for idx, text in enumerate(test):
      test_tagged.append(TaggedDocument(text, tags=[str(idx)]))
  vectors = model.infer_vector(test_tagged)
except:
  print("Evaluation pipeline for main model that may need additional preprocessing of te
```

```
Evaluation pipeline for main model that may need additional preprocessing of test IDs if
error running
```

Results and Analyses

Discussion

To do before final project submission

PLANS

Currently, the codebase of my project is undergoing a significant refactoring to improve maintainability and performance while handling the intricacies of working with both the demo and actual data provided by the MIMIC-III dataset.

Although the complete workflow from training to inference is conceptualized in a draft form, it has not yet been fully implemented. The forthcoming stages of my project will involve training the model on the preprocessed datasets, followed by testing to evaluate the model's efficacy.

This will be pivotal in validating our hypothesis regarding the model's performance and potential improvements over existing methods. Upon completion of these steps, I will comprehensively analyze the results and document my findings and conclusions in a detailed report.

REFERENCES

1. Zhang, D., Yin, C., Zeng, J. et al. Combining structured and unstructured data for predictive models: a deep learning approach. BMC Med Inform Decis Mak 20, 280 (2020). https://doi.org/10.1186/s12911-020-01297-6

2. Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). Dive into deep learning. Cambridge University Press.

```
In [30]:
```

```
In [30]:
```