**Prediction of ICU Readmissions**

Unplanned readmissions to ICU contribute to high health care costs and poor patient outcomes. 6-7% of all ICU cases see a readmission within 72 hours. Machine learning models on electronic health record data can help identify these cases, providing more information about short and long-term risks to clinicians at the time of ICU discharge

1. Topic

In this project I’ll try to explore how to predict ICU readmission based on publicly available dataset (MIMIC) and optimize the model to get a reasonable performance

1. Dataset

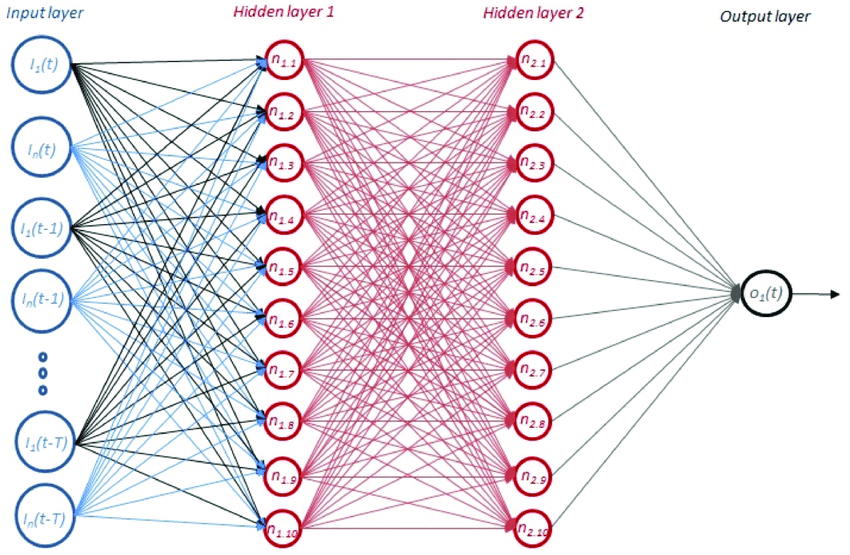
In MIMIC III dataset, (<https://mimic.physionet.org/>), we have a registry developed by the MIT Lab for Computational Physiology, comprising deidentified health data associated with ~40,000 critical care patients. It includes demographics, vital signs, laboratory tests, medications, and more.

In my github page (<https://github.com/ZepengHuo/deep_learning_class.git>), you can find the extracted data, stored in csv file format, named as df\_MASTER\_DATA.csv. It has 2354 features, ranging from heart rate, blood pressure, etc.

In the dataset, you can find out whether a patient has been readmitted in 72 hours, and I’ll use that as my training labels

1. Neural network model

I used grid search to search all the optimal hyper-parameters (e.g. number of neurons) to train a neural network, code is ([GridSearch\_NN.py](https://github.com/ZepengHuo/deep_learning_class/blob/master/GridSearch_NN.py)), result is ([Grid\_search\_results.png](https://github.com/ZepengHuo/deep_learning_class/blob/master/Grid_search_results.png)).

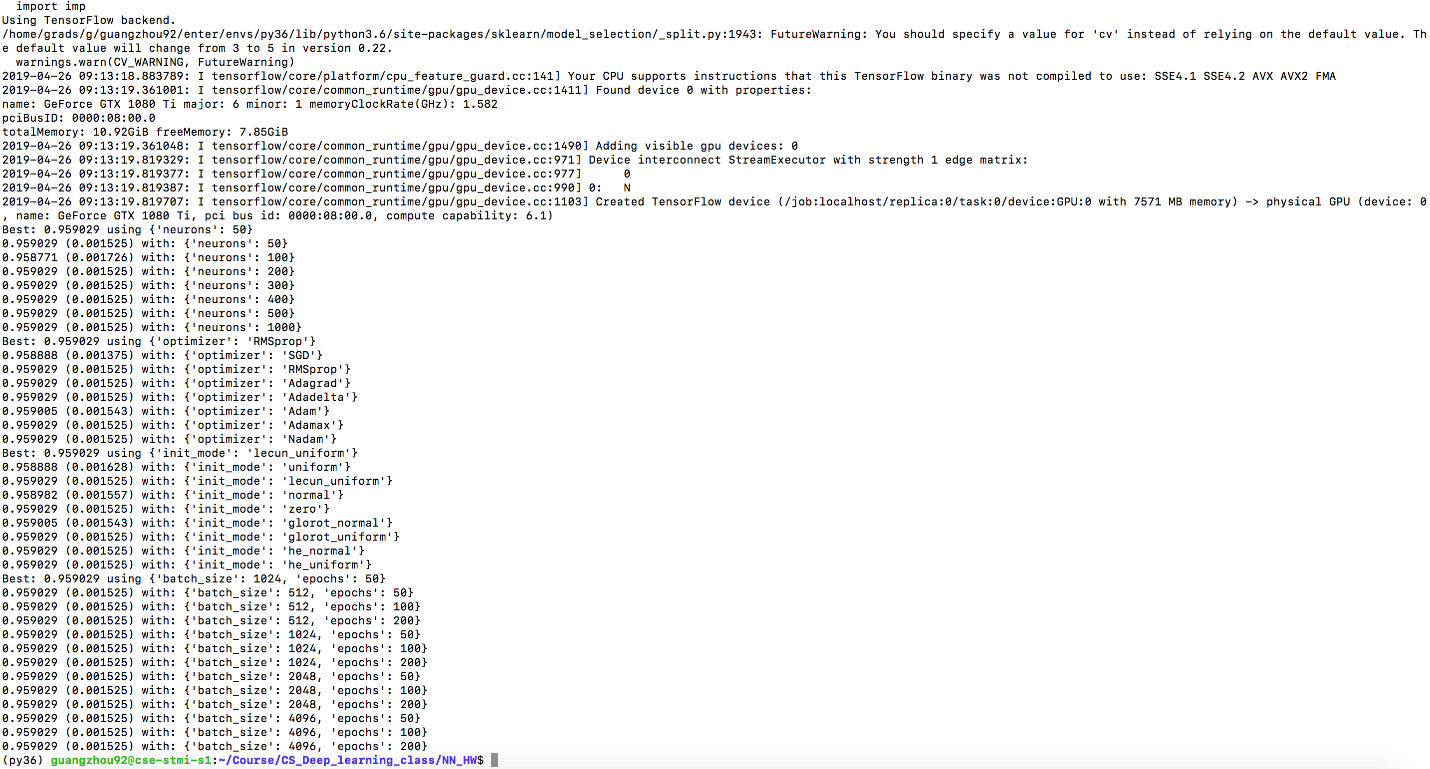


100 neurons

2353 input dimension

50 neurons

As shown in the figure, I have input variables from the left and one output (readmission or not) on the right.



The grid search result is shown above. And I used the best of each hyper-parameter to train the neural network. The architecture is dense network

X\_train is 42664 rows × 2353 columns

X\_test is 10207 rows × 2353 columns

Input layer: 2353

Hidden layer 1:

* 50 neurons
* Shape of tensor: 2353 \* 50

Hidden layer 2:

* 100 neurons
* Shape of tensor: 50 \* 100

Loss function: 'binary\_crossentropy'

Output layer: 1 neuron

1. Annotated code

|  |
| --- |
| import os  os.environ["CUDA\_DEVICE\_ORDER"]="PCI\_BUS\_ID"  device\_nb = 2  os.environ["CUDA\_VISIBLE\_DEVICES"]=str(device\_nb)  import pandas as pd  import datetime  from xgboost import XGBClassifier  from sklearn.model\_selection import GridSearchCV  import tensorflow as tf  from keras import backend as K  import tarfile  import numpy as np  import \_pickle as cPickle  import os  import wfdb  from datetime import datetime  from datetime import timedelta  import numpy  from sklearn.model\_selection import GridSearchCV  from keras.models import Sequential  from keras.layers import Dense  from keras.layers import Dropout  from keras.wrappers.scikit\_learn import KerasClassifier  from keras.constraints import maxnorm  from sklearn.model\_selection import KFold  import numpy as np  from sklearn.metrics import roc\_auc\_score  from sklearn.metrics import make\_scorer  df\_o = pd.read\_csv("df\_MASTER\_DATA.csv")  import random  random.seed(9)  random.sample(range(1, 61089), 10000)  #sub-select 20% for testing  test\_idx = random.sample(range(df\_o.shape[0]), int(df\_o.shape[0] \* 0.2))  df\_test = df\_o.loc[test\_idx]  train\_idx = list(set([ x for x in range(df\_o.shape[0])]) - set(test\_idx))  df\_train = df\_o.loc[train\_idx]  best\_neurons = 50  best\_optimizer = 'RMSprop'  best\_init\_mode = 'lecun\_uniform'  best\_batch\_size = 1024  best\_epochs = 50  def create\_model(neurons=best\_neurons, optimizer=best\_optimizer, init\_mode=best\_init\_mode, batch\_size=best\_batch\_size,  epochs=best\_epochs):  # create model  model = Sequential()  model.add(Dense(neurons, input\_dim=2291, activation='relu',  kernel\_initializer=init\_mode, kernel\_constraint=maxnorm(4)))  model.add(Dropout(0.2))  model.add(Dense(1, kernel\_initializer=init\_mode, activation='sigmoid'))  # Compile model  model.compile(loss='binary\_crossentropy', optimizer=best\_optimizer, metrics=['accuracy'])  #model.compile(loss='binary\_crossentropy', optimizer='adam')  return model  #mean impute  #delete HADM\_ID, SUBJECT\_ID, ETHNICITY, MARITAL\_STATUS, INSURANCE, RELIGION, INTIME, OUTTIME  #dummify  def preprocessing(df\_labeled):  #pre-processing, missingness > 0.5  df\_labeled\_filtered = df\_labeled  for column in df\_labeled\_filtered.columns:  missing\_r = df\_labeled[column].isnull().sum()/df\_labeled.shape[0]  if missing\_r > 0.5:  #keep arterial BP  if not (column.startswith('Arterial\_BP') or column.startswith('Mean\_Arterial')):  df\_labeled\_filtered = df\_labeled\_filtered.drop(columns=[column])  #mean impute  df\_labeled\_filtered = df\_labeled\_filtered.fillna(df\_labeled\_filtered.mean())  #delete HADM\_ID, SUBJECT\_ID, ETHNICITY, MARITAL\_STATUS, INSURANCE, RELIGION, INTIME, OUTTIME  colmuns\_todrop = ['Unnamed: 0','FIRST\_CAREUNIT','ETHNICITY','MARITAL\_STATUS','HADM\_ID', 'ICUSTAY\_ID',  'INSURANCE','RELIGION','INTIME', 'OUTTIME', 'SUBJECT\_ID','LANGUAGE','Time\_To\_readmission',  'IsReadmitted\_24hrs', 'IsReadmitted\_48hrs', 'IsReadmitted\_7days',  'IsReadmitted\_30days', 'IsReadmitted\_Bounceback']  #don't delete ==> 'IsReadmitted\_72hrs',    for column\_d in colmuns\_todrop:    try:  df\_labeled\_filtered = df\_labeled\_filtered.drop(columns=column\_d)  except:  pass  #dummify  df\_labeled\_filtered = pd.get\_dummies(df\_labeled\_filtered)      return df\_labeled\_filtered  def fivefold\_auroc(df\_labeled\_filtered):  #Up- and down-sample for imbalance class  X\_1 = df\_labeled\_filtered[df\_labeled\_filtered['IsReadmitted\_72hrs'] == 1]  X\_0 = df\_labeled\_filtered[df\_labeled\_filtered['IsReadmitted\_72hrs'] == 0].iloc[:X\_1.shape[0], :]    X\_all = X\_0.append(X\_1)    X\_all = df\_labeled\_filtered  y = X\_all['IsReadmitted\_72hrs']  X = X\_all.drop(columns=['IsReadmitted\_72hrs'])  #X = df\_labeled\_filtered.drop(columns=['label'])  #y = df\_labeled\_filtered['label']  kf = KFold(n\_splits=5, shuffle=True)  kf.get\_n\_splits(X)  fold\_num = 1  for train\_index, test\_index in kf.split(X):  #print("TRAIN:", train\_index, "TEST:", test\_index)  model = KerasClassifier(build\_fn=create\_model, epochs=best\_epochs, batch\_size=best\_batch\_size, verbose=0)  model.fit(X.iloc[train\_index], y.iloc[train\_index])  print('Fold %d' %fold\_num, end=' ')  fold\_num += 1  print(roc\_auc\_score(y.iloc[test\_index], model.predict\_proba(X.iloc[test\_index])[:,1]))  df\_filtered = preprocessing(df\_train)  fivefold\_auroc(df\_filtered) |

I have the code to run a 5 fold cross validation, and iterate through each of them, each time a 20% of data will be used as testing set, and the remaining will be used as training, until all data has been used.

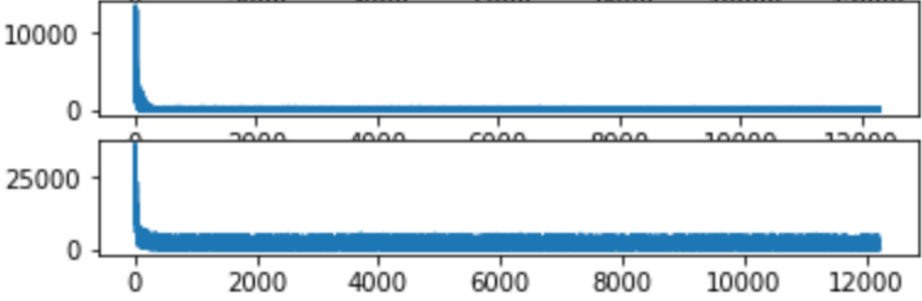
1. Visualization/demonstration

Run the commend in terminal:

**python medical\_prediction.py**

And you will see input and output window. In my project, I’m planning to use medical data registry to  
predict whether the patient will be readmitted to an ICU (a minor revision from last video). Since  
readmission to an ICU in a short time will incur unnecessary overhead and potentially influence the  
mortality rate. So if the model can predict if a patient will be readmitted at what time, it can largely  
reduce the patient risk and increase survival rate  
The demo will first require you generate test patient data from a test set (data here is a mockup), and  
then it will predict the readmission for those patients. Lastly it will plot the result of how well the model  
predicts.

Github link: https://github.com/ZepengHuo/deep\_learning\_class.git  
Video link: <https://youtu.be/KkjTmxNwnSs>



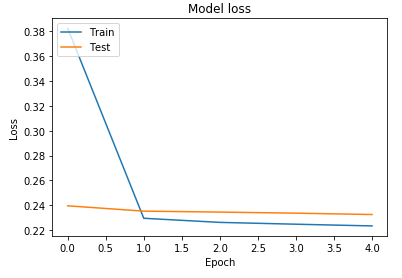
Training loss

Validation loss

Epochs

For improvement, I’ve trained a LSTM model on only the time-series data in the MIMIC dataset

The loss look like this:



However, when I jointly train them together, by concatenation of LSTM to the dense network, the performance didn’t improve much, since the original accuracy is already really high (~0.96). So I concluded the dense network can already prediction readmission in an ideal performance. The code for LSTM implementation:

|  |
| --- |
| # coding: utf-8  # In[8]:  import pandas as pd  import datetime  from xgboost import XGBClassifier  from sklearn.model\_selection import GridSearchCV  import tarfile  import numpy as np  import \_pickle as cPickle  import os  import wfdb  from datetime import datetime  from datetime import timedelta  from keras.models import Sequential  from keras.layers import LSTM  from keras.layers import Dense  from keras import backend as K  # In[2]:  #df\_CE = pd.read\_csv('all\_chart\_events.csv')  #df\_CE.columns = ['icustay\_id', 'itemid', 'valuenum', 'charttime']  # In[3]:  #df\_itemID = pd.read\_csv('unique\_chartevent\_items.csv')  #df\_itemID.columns = ['itemid', 'measurement']  # In[9]:  df\_trainID = pd.read\_csv('df\_train\_subjects.csv')  df\_testID = pd.read\_csv('df\_test\_subjects.csv')  df\_valID = pd.read\_csv('df\_val\_subjects.csv')  # In[10]:  df\_trainID  # # bounce back has 6% of the data  # In[5]:  import os  path, dirs, files = next(os.walk("./each\_icustay\_csv"))  file\_count = len(files)  file\_count  # In[6]:  import os  path, dirs, files = next(os.walk("./feature\_folder"))  file\_count = len(files)  file\_count  # # feature space has non-sparse ratio mean: 23%, max 51%, min 0%  # # get features prepared for LSTM (training set)  # In[22]:  #nan is from mean, std, that there's no mean, std (just one value in a window)  #0 is from max, min, count  x\_train = []  y\_train = []  for index, row in df\_trainID.iterrows():    icustay\_id = row['ICUSTAY\_ID']      #see if a icu stay has a feature file  try:  outfile = './feature\_folder/feature\_space\_%s.file.npy'%(str(icustay\_id))  all\_feature\_vectors = np.load(outfile)  #change NaN to 0  all\_feature\_vectors = np.nan\_to\_num(all\_feature\_vectors)      #count non-zero  #non\_0 = np.count\_nonzero(np.nan\_to\_num(all\_feature\_vectors))  #non\_0\_ratio = non\_0/(all\_feature\_vectors.shape[0]\*all\_feature\_vectors.shape[1])  x\_train.append(all\_feature\_vectors)  y\_train.append(row['IsReadmitted\_Bounceback'])    except FileNotFoundError:  continue    x\_train = np.asarray(x\_train)  y\_train = np.asarray(y\_train)  # In[24]:  x\_train.shape  # # get features prepared for LSTM (validation set)  # In[8]:  #nan is from mean, std, that there's no mean, std (just one value in a window)  #0 is from max, min, count  x\_valid = []  y\_valid = []  for index, row in df\_valID.iterrows():    icustay\_id = row['ICUSTAY\_ID']      #see if a icu stay has a feature file  try:  outfile = './feature\_folder/feature\_space\_%s.file.npy'%(str(icustay\_id))  all\_feature\_vectors = np.load(outfile)  #change NaN to 0  all\_feature\_vectors = np.nan\_to\_num(all\_feature\_vectors)      #count non-zero  #non\_0 = np.count\_nonzero(np.nan\_to\_num(all\_feature\_vectors))  #non\_0\_ratio = non\_0/(all\_feature\_vectors.shape[0]\*all\_feature\_vectors.shape[1])  x\_valid.append(all\_feature\_vectors)  y\_valid.append(row['IsReadmitted\_Bounceback'])    except FileNotFoundError:  continue    x\_valid = np.asarray(x\_valid)  y\_valid = np.asarray(y\_valid)  # # get features prepared for LSTM (testing set)  # In[9]:  #nan is from mean, std, that there's no mean, std (just one value in a window)  #0 is from max, min, count  x\_test = []  y\_test = []  for index, row in df\_testID.iterrows():    icustay\_id = row['ICUSTAY\_ID']      #see if a icu stay has a feature file  try:  outfile = './feature\_folder/feature\_space\_%s.file.npy'%(str(icustay\_id))  all\_feature\_vectors = np.load(outfile)  #change NaN to 0  all\_feature\_vectors = np.nan\_to\_num(all\_feature\_vectors)      #count non-zero  #non\_0 = np.count\_nonzero(np.nan\_to\_num(all\_feature\_vectors))  #non\_0\_ratio = non\_0/(all\_feature\_vectors.shape[0]\*all\_feature\_vectors.shape[1])  x\_test.append(all\_feature\_vectors)  y\_test.append(row['IsReadmitted\_Bounceback'])    except FileNotFoundError:  continue    x\_test = np.asarray(x\_test)  y\_test = np.asarray(y\_test)  # # test: not output along time dimension, but along neurons dimesnion  # In[17]:  import os  os.environ["CUDA\_DEVICE\_ORDER"]="PCI\_BUS\_ID"  device\_nb = '7'  os.environ["CUDA\_VISIBLE\_DEVICES"]=str(device\_nb)  from keras.models import Model  from keras.layers import Input  from keras.layers import LSTM, Dense  from numpy import array  # define model  inputs1 = Input(shape=(3, 1))  #lstm1 = LSTM(2, return\_state=True)(inputs1)  lstm1 = LSTM(10, name='lstm')(inputs1)  output = Dense(1)(lstm1)  model = Model(inputs=inputs1, outputs=output)  # define input data  data = array([[0.8, 0.9, 0.7],[0.1, 0.2, 0.3]]).reshape((2,3,1))  # make and show prediction  prediction=model.predict(data)  print(prediction)  # In[18]:  df = pd.DataFrame(data.reshape(2,-1))  df.values  # In[6]:  model.summary()  # In[20]:  layer\_name = 'lstm'  intermediate\_layer\_model\_output = Model(inputs=model.input,  outputs=model.get\_layer(layer\_name).output)  intermediate\_output = intermediate\_layer\_model\_output.predict(df.values.reshape(2,3,1))  # In[21]:  intermediate\_output  # In[25]:  pd.DataFrame(intermediate\_output)  # In[2]:  from numba import cuda  cuda.select\_device(0)  cuda.close()  # In[26]:  cohortvector\_train = pd.read\_csv('cohortvector\_train.csv')  # In[27]:  cohortvector\_train  # # LSTM  # In[10]:  import os  os.environ["CUDA\_DEVICE\_ORDER"]="PCI\_BUS\_ID"  device\_nb = 2  os.environ["CUDA\_VISIBLE\_DEVICES"]=str(device\_nb)  model = Sequential()  #model.add(LSTM(32, return\_sequences=True, input\_shape=(48, 30)))  model.add(LSTM(2048, input\_shape=(48, 30)))  model.add(Dense(1, activation='sigmoid'))  model.compile(loss='binary\_crossentropy', optimizer='adam')  history = model.fit(x\_train, y\_train, epochs=5, batch\_size=2014, verbose=2, validation\_data=(x\_valid, y\_valid))  y\_pred\_proba = model.predict\_proba(x\_test)  y\_pred = model.predict(x\_test)  # In[23]:  from sklearn.metrics import roc\_auc\_score  roc\_auc\_score(y\_test, y\_pred\_proba)  # In[12]:  import matplotlib.pyplot as plt  # Plot training & validation loss values  plt.plot(history.history['loss'])  plt.plot(history.history['val\_loss'])  plt.title('Model loss')  plt.ylabel('Loss')  plt.xlabel('Epoch')  plt.legend(['Train', 'Test'], loc='upper left')  plt.show()  # In[22]:  y\_pred\_proba  # In[21]:  y\_pred  # In[6]:  from numba import cuda  cuda.select\_device(0)  cuda.close()  # In[15]:  itemid1 = 211  itemid2 = 224697  icustay\_id = 294638  intime = pd.to\_datetime(df\_trainID[df\_trainID['ICUSTAY\_ID'] == icustay\_id]['INTIME'].values[0])  outtime = pd.to\_datetime(df\_trainID[df\_trainID['ICUSTAY\_ID'] == icustay\_id]['OUTTIME'].values[0])  with open("./each\_icustay\_csv/" + str(icustay\_id) + '.csv') as f0:  Adf\_ICUstay\_CE = pd.read\_csv(f0)  onetime = pd.to\_datetime(Adf\_ICUstay\_CE[(Adf\_ICUstay\_CE['itemid'] == itemid1) | (Adf\_ICUstay\_CE['itemid'] == itemid2)]['charttime']).values[0]  df\_onechannel = Adf\_ICUstay\_CE[(Adf\_ICUstay\_CE['itemid'] == itemid1) | (Adf\_ICUstay\_CE['itemid'] == itemid2)]  #Adf\_ICUstay\_CE[(Adf\_ICUstay\_CE['itemid'] == itemid1) | (Adf\_ICUstay\_CE['itemid'] == itemid2)]  # In[16]:  pd.to\_datetime(df\_onechannel['charttime'])  df\_onechannel['charttime'] = pd.to\_datetime(df\_onechannel['charttime'])  df\_onechannel['charttime'] .values[0]  # In[17]:  #given a icu stay's df, and what itemid, to get features in 48h  def OnefeaturesIN48h(Adf\_ICUstay\_CE, itemid1, itemid2, intime, outtime):    feature\_vector = []    df\_onechannel = Adf\_ICUstay\_CE[(Adf\_ICUstay\_CE['itemid'] == itemid1) | (Adf\_ICUstay\_CE['itemid'] == itemid2)]    #change 'charttime' str to datetime object  df\_onechannel['charttime'] = pd.to\_datetime(df\_onechannel['charttime'])    #TO DO: first impute all of them    end\_time = intime    #first 24 hours  for pasthours in range(0, 24):      start\_time = end\_time  end\_time = start\_time + np.timedelta64(1, 'h')    df\_InATimewindow = df\_onechannel[ (start\_time < df\_onechannel['charttime']) & (df\_onechannel['charttime'] < end\_time) ]      #add mean feature  try:  mean\_feature = np.mean(df\_InATimewindow['valuenum'].values)  except ValueError:  mean\_feature = 0      #add std feature  try:  std\_feature = np.std(df\_InATimewindow['valuenum'].values)  except ValueError:  std\_feature = 0    #add max feature  try:  max\_feature = np.amax(df\_InATimewindow['valuenum'].values)  except ValueError:  max\_feature = 0    #add min feature  try:  min\_feature = np.amin(df\_InATimewindow['valuenum'].values)  except ValueError:  min\_feature = 0    #add count feature  try:  count\_feature = df\_InATimewindow['valuenum'].values.shape[0]  except ValueError:  count\_feature = 0    features\_1h\_window = [mean\_feature, std\_feature, max\_feature, min\_feature, count\_feature]    feature\_vector.append(features\_1h\_window)        end\_time = outtime - np.timedelta64(24, 'h')    #last 24 hours  for pasthours in range(0, 24):    start\_time = end\_time  end\_time = start\_time + np.timedelta64(1, 'h')    df\_InATimewindow = df\_onechannel[ (start\_time < df\_onechannel['charttime']) & (df\_onechannel['charttime'] < end\_time) ]      #add mean feature  try:  mean\_feature = np.mean(df\_InATimewindow['valuenum'].values)  except ValueError:  mean\_feature = 0      #add std feature  try:  std\_feature = np.std(df\_InATimewindow['valuenum'].values)  except ValueError:  std\_feature = 0    #add max feature  try:  max\_feature = np.amax(df\_InATimewindow['valuenum'].values)  except ValueError:  max\_feature = 0    #add min feature  try:  min\_feature = np.amin(df\_InATimewindow['valuenum'].values)  except ValueError:  min\_feature = 0    #add count feature  try:  count\_feature = df\_InATimewindow['valuenum'].values.shape[0]  except ValueError:  count\_feature = 0      features\_1h\_window = [mean\_feature, std\_feature, max\_feature, min\_feature, count\_feature]    feature\_vector.append(features\_1h\_window)        return feature\_vector  # In[18]:  def AICUstay\_goodchannels(df\_CE, df\_itemID, icustay\_id):  goodchannels = []  for item\_count in df\_CE[df\_CE['icustay\_id'] == icustay\_id]['itemid'].value\_counts().iteritems():  itemid = item\_count[0]  item\_name = df\_itemID[df\_itemID['itemid']==itemid]['measurement'].values[0]  goodratio = missing\_ratio(df\_CE, icustay\_id, itemid)  if goodratio >= 0.5:  goodchannels.append(item\_name)  #print(item\_name, goodratio)  return goodchannels  # In[19]:  def missing\_ratio(df\_CE, icustay\_id, itemid):  return (df\_CE[(df\_CE['icustay\_id'] == icustay\_id) & (df\_CE['itemid'] == itemid)]['valuenum'].shape[0]-df\_CE[(df\_CE['icustay\_id'] == icustay\_id) & (df\_CE['itemid'] == itemid)]['valuenum'].isnull().sum()) / df\_CE[(df\_CE['icustay\_id'] == icustay\_id) & (df\_CE['itemid'] == itemid)]['valuenum'].shape[0] |