A2_part1_LSTM_on_EHR_structured

November 2, 2022

1 Assignment 2 - clinical prediction with LSTMs on structured MIMIC-III data

The Intensive Care Unit (ICU) treats, on estimate, 55,000 patients per day. ICU patients have an average length of stay of 3.8 days with a mortality rate of 10-29% soucre. Furthermore, monitoring patients in ICU rooms requires keeping track of tremendous amounts of real-time information, and much of it is logged and stored in electronic health record (EHR) systems. Information overload can be seen as a huge barrier to safe and efficient healthcare delivery. As such, there has been much work on exploring computer assisted diagnostic (CAD) systems to predict clinical outcomes from these data sources.

In the first part of this assignment, you'll be working with structured data measurements (e.g. heart rate, glucose levels, central venous pressure) to make predictions of - sepsis - myocardial infarction (MI) - vancomycin antibiotic administration

over two week patient ICU courses. We'll be running a simplified version of the models in An attention based deep learning model of clinical events in the intensive care unit. (In parts 2 & 3 of this assignment you'll use unstructurd clinical text from discharge summaries.)

Q1.1 clinical applications of prediction models

Many researchers work on the problem of predicting sepsis. Briefly explain how a sepsis prediction model can improve clinical outcomes for patients.

1.0.1 Written answer: Sepsis is the body's extreme response to an infection. It is a life-threatening medical emergency and without timely treatment, sepsis can rapidly lead to tissue damage, organ failure, and death. Treating sepsis requires antibiotic treatments and IVs as soon as possible. A sepsis prediction model could greately improve clinical outcomes for patients since diagnosing and treating sepsis is very time sensitive. If one were able to predict sepsis onset in advance, a clinician can quickly begin antibiotic treatment before sepsis becomes life threatening or tissue damaging.

1.1 MIMIC-III Data Preprocessing & Visualization

In the below cell, define the path to `ROOT`. This is where all assignment 2 data will be placed. E.g. you could put it in the same directory as the notebook.

Go to https://physionet.org/sign-dua/mimiciii/1.4/ and accept the MIMIC-III data use agreement.

Navigate to `ROOT` in your terminal, and then download the MIMIC-III dataset by executing the following commands (replace username with your physionet username). This will create a directory in ROOT called `mimic_database/`.

`wget -r -N -c -np --user <username> --ask-password https://physionet.org/files/mimiciii/1.4/` (2-5 minutes)

`mkdir -p mimic_database && mv physionet.org/files/mimiciii/1.4/*.csv.gz mimic_database/ && rm -rf physionet.org/ && cd mimic_database && gunzip *.gz && echo 'Success!' || echo 'Failure'` (5-10 mins)

If everything worked, the final output should read `Success!`, and `mimic_database/` should contain some csv files.

[40]:
```python
import pickle
import math
import re
import csv
import concurrent.futures
import os
from functools import reduce
import pathlib
import pickle

from operator import add
import pandas as pd
import numpy as np

import gc
from time import time
import math
import pickle
import pathlib
import matplotlib.pyplot as plt

import tensorflow as tf

# packages from current directory
import parser_utils
import data_utils

# config
tf.keras.backend.set_floatx("float32")

ROOT = "/home/jupyter/cs271_assign2/ROOT"  # Put your root path here
```

(~40 mins) Run the next cell after setting `DO_PARSING` and `DO_BUILD_DATASETS` to True. It's slow, but only needs to be run once. It will create files in `ROOT/mapped_events/`, and in `ROOT/saved_data`, and we'll explain what it's doing a bit later.

Once it runs successfully, set `DO_PARSING` and `DO_BUILD_DATASETS` to False.

```
[41]: DO_PARSING = False
      DO_BUILD_DATASETS = False

      if DO_PARSING:
          parser_utils.do_all_parsing(ROOT, verbose=1)
      if DO_BUILD_DATASETS:
          data_utils.build_seq_datasets(ROOT)
```

Q1.2 mimic database exploration I

Let's get a better understanding of the MIMIC-III database. Here is the documentation. The 'Data Description' section is especially useful. When you ran the data dowload commands at the start of this notebook (the command starting with `wget`), a directory was created in ROOT called `mimic_databse/`. This is contains the MIMIC csv files.

First load the `PATIENTS.csv` file and display the results to screen (hint: use Pandas to load the csv's to a DataFrame; hint 2: the function `display(df)` prints the dataframes nicely).

```
[42]: # YOUR CODE HERE #
      path = "/home/jupyter/cs271_assign2/ROOT/mimic_database/PATIENTS.csv"
      patients = pd.read_csv(path)
      display(patients)
      # END CODE #
```

```
       ROW_ID  SUBJECT_ID GENDER                  DOB                  DOD  \
0         234         249      F  2075-03-13 00:00:00                  NaN
1         235         250      F  2164-12-27 00:00:00  2188-11-22 00:00:00
2         236         251      M  2090-03-15 00:00:00                  NaN
3         237         252      M  2078-03-06 00:00:00                  NaN
4         238         253      F  2089-11-26 00:00:00                  NaN
…         …           …    …                    …                    …
46515   31840       44089      M  2026-05-25 00:00:00                  NaN
46516   31841       44115      F  2124-07-27 00:00:00                  NaN
46517   31842       44123      F  2049-11-26 00:00:00  2135-01-12 00:00:00
46518   31843       44126      F  2076-07-25 00:00:00                  NaN
46519   31844       44128      M  2098-07-25 00:00:00                  NaN

                  DOD_HOSP DOD_SSN  EXPIRE_FLAG
0                      NaN     NaN            0
1      2188-11-22 00:00:00     NaN            1
2                      NaN     NaN            0
3                      NaN     NaN            0
4                      NaN     NaN            0
…                      …       …            …
46515                  NaN     NaN            0
46516                  NaN     NaN            0
46517  2135-01-12 00:00:00     NaN            1
46518                  NaN     NaN            0
46519                  NaN     NaN            0
```

```
[46520 rows x 8 columns]
```

Notice that the date of birth (DOB) and date of death (DOD) are in the future. Briefly explain why (hint: see Methods section of MIMIC documentation).

1.1.1 Written answer: According to the MIMIC documentation, "dates were shifted into the future by a random offset for each individual patient in a consistent manner to preserve intervals, resulting in stays which occur sometime between the years 2100 and 2200". The HIPAA deidentification process for structured data requires the removal of all eighteen identifying data elements (listed in HIPAA). Two of these identifying data elements are DOB and DOD, so they were randomly shifted to ensure anonymity.

Q1.3 mimic database exploration II

In the next code cell, use the dataframe from `PATIENTS.csv` to print the following summary measurements about the dataset: - The number of total patients. - The counts of male and female patients. - The count of patients with a death on record.

(Hint: the `groupby()` function may be useful).

[43]:
```
# YOUR CODE HERE #
total_patients = len(np.unique(patients["SUBJECT_ID"]))
print("Total number of patients = {}.".format(total_patients))
gender = patients.groupby(['GENDER']).agg("count")["SUBJECT_ID"]
print("Count of female patients = {}. Count of male patients = {}.".
  ↪format(gender['F'], gender['M']))
deaths = patients['DOD'].count()
print("Count of patients with death on record = {}.".format(deaths))
# END CODE #
```

```
Total number of patients = 46520.
Count of female patients = 20399. Count of male patients = 26121.
Count of patients with death on record = 15759.
```

Q1.4 mimic database exploration III

Let's now look at `CHARTEVENTS.csv`, which has one row for each recorded chart measurement (e.g. features like heart rate, glucose levels, central venous pressure). This file is 33GB so don't try to load the whole thing into memory.

Read the first 100 rows of the file `CHARTEVENTS.csv` and display the DataFrame in the notebook.

[44]:
```
first_nrows = 100
path = "/home/jupyter/cs271_assign2/ROOT/mimic_database/CHARTEVENTS.csv"
# YOUR CODE HERE #
chartevents = pd.read_csv(path, nrows = first_nrows)
display(chartevents)
# END CODE #
```

```
   ROW_ID  SUBJECT_ID  HADM_ID  ICUSTAY_ID  ITEMID            CHARTTIME  \
0     788          36   165660      241249  223834  2134-05-12 12:00:00
1     789          36   165660      241249  223835  2134-05-12 12:00:00
2     790          36   165660      241249  224328  2134-05-12 12:00:00
3     791          36   165660      241249  224329  2134-05-12 12:00:00
4     792          36   165660      241249  224330  2134-05-12 12:00:00
..     …         …       …          …      …                   …
95    348          34   144319      290505  226873  2191-02-23 07:31:00
96    349          34   144319      290505  220210  2191-02-23 07:33:00
97    350          34   144319      290505  220045  2191-02-23 07:34:00
98    351          34   144319      290505  220179  2191-02-23 07:34:00
99    352          34   144319      290505  220180  2191-02-23 07:34:00

             STORETIME   CGID   VALUE  VALUENUM  VALUEUOM  WARNING  ERROR  \
0   2134-05-12 13:56:00  17525   15.00     15.00     L/min        0      0
1   2134-05-12 13:56:00  17525  100.00    100.00       NaN        0      0
2   2134-05-12 12:18:00  20823    0.37      0.37       NaN        0      0
3   2134-05-12 12:19:00  20823    6.00      6.00       min        0      0
4   2134-05-12 12:19:00  20823    2.50      2.50       NaN        0      0
..                    …     …     …       …        …       …     …
95  2191-02-23 07:35:00  16924    1.00      1.00       NaN        0      0
96  2191-02-23 07:45:00  17741   26.00     26.00  insp/min        0      0
97  2191-02-23 10:53:00  17741   44.00     44.00       bpm        0      0
98  2191-02-23 07:45:00  17741  135.00    135.00      mmHg        0      0
99  2191-02-23 07:45:00  17741   61.00     61.00      mmHg        0      0

   RESULTSTATUS  STOPPED
0           NaN      NaN
1           NaN      NaN
2           NaN      NaN
3           NaN      NaN
4           NaN      NaN
..          …       …
95          NaN      NaN
96          NaN      NaN
97          NaN      NaN
98          NaN      NaN
99          NaN      NaN

[100 rows x 15 columns]
```

Each row is a single measurement, but it does not say what is being measured. Explain how you could find this out. (Hint: see the ‘Data Description’ section of the documentation.)

1.1.2 Written answer: According to the MIMIC-III documentation, every row of CHARTEVENTS is associated with an ITEMID which represents the concept being measured, but does not give the name of the measurement. However, by joining CHARTEVENTS and D_ITEMS on ITEMID, you can uncover the concept represented by a given ITEMID.

So far we've looked at the original MIMIC-III database, but it's not in a format suitable for a sequence model prediction (like LSTMs or transformers). You ran two lines of code at the start of the assignment to get it in the right format.

The first function was `parser_utils.do_all_parsing`, which created a set of files in `ROOT/mimic_database/mapped_events`, which are closer to what we need. One output was the file `CHARTEVENTS_reduced_24_hour_blocks_plus_admissions_plus_patients_plus_scripts_plus_icds_plus_n` This file: - Is similar to `CHARTEVENTS`, except that each measurement is **assigned to a single 24hr block** (like 2117-09-11), rather than a specific timestamp (like 2117-09-11 16:04:00). You can think of this as discretizing the dataset. - Each row also has extra data about the patient: admission times, scripts, and known patient diseases (ICD's).

Next you ran `data_utils.load_seq_dataset` which does the following: - Given a prediction target (one of `MI`, `SEPSIS`, or `VANCOMYCIN`), generate numpy arrays for the train, test, and validation set. The data is shuffled before splitting. - Put X-data into the shape `(n_hostpital_stays, n_timesteps, n_features)`. So `X[i,j,k]` gives the kth feature, for the jth day of the ith hospital stay. - Puts the labels, y-data, into shape `(n_hospital_stays,n_timesteps,1)`. All 3 prediction problems are binary, to these values 0 or 1. - Returns a list of strings called `features` containing the names of each feature in the X-data. So `features[k]` is the name of the features in `X[:,:,k]` - Zero-padding. Since not all patients will have a valid measurememt for every feature at each timestep, we fill the remainder with zeros. Later we will tell the model to ignore this data in training. - Z-score normalization.

Now we can load that data using the following function call:

[45]:
```python
target = "SEPSIS"  # 'SEPSIS' or 'MI' or 'VANCOMYCIN'
(
    train_x,
    val_x,
    train_y,
    val_y,
    no_feature_cols,
    test_x,
    test_y,
    x_boolmat_test,
    y_boolmat_test,
    x_boolmat_val,
    y_boolmat_val,
    features,
) = data_utils.load_seq_dataset(ROOT, target)

# convert all float64 to float32
train_x = train_x.astype(np.float32)
```

```
val_x = val_x.astype(np.float32)
test_x = test_x.astype(np.float32)
train_y = train_y.astype(np.float32)
val_y = val_y.astype(np.float32)
test_y = test_y.astype(np.float32)

print("train shapes ", train_x.shape, train_y.shape)
print("val shapes   ", val_x.shape, val_y.shape)
print("test shapes  ", test_x.shape, test_y.shape)
print("# features   ", len(features))
```

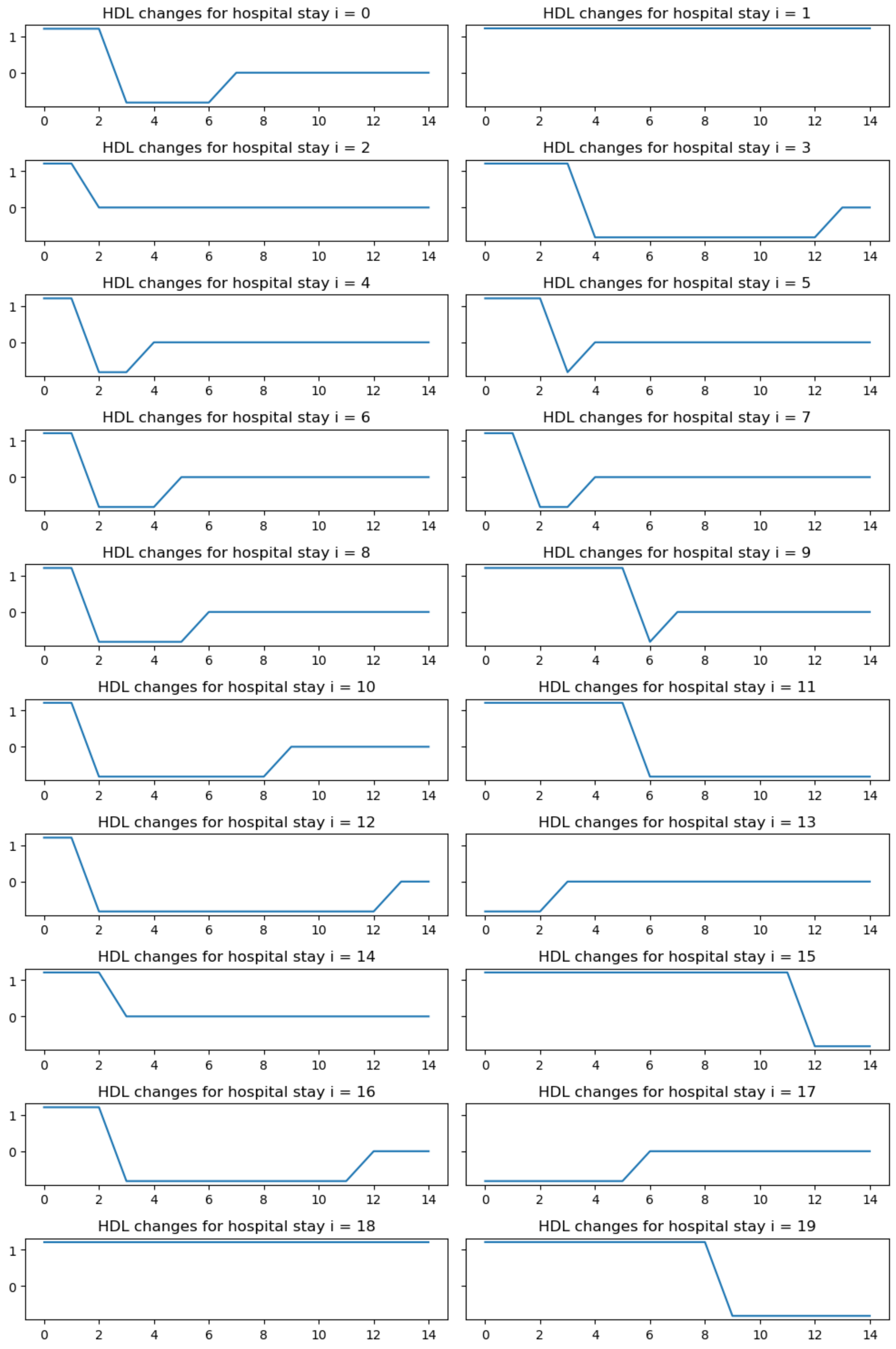
```
train shapes  (2906, 15, 226) (2906, 15, 1)
val shapes    (5178, 15, 226) (5178, 15, 1)
test shapes   (10355, 15, 226) (10355, 15, 1)
# features    226
```

Q1.5 sequence model data sets

Let's look at some samples from `train_x`. In the below code cell, get the feature called 'HDL', and generate 20 plots showing how this variable changes over all the timesteps for the first 20 hospital stays (1 time series plot per hospital stay).

[49]:
```
# YOUR CODE HERE #
idx = features.index('HDL')
n_hospital_stays, n_timesteps, n_features = train_x.shape
fig, axs = plt.subplots(10, 2, figsize=(10,15), sharey=True)
for i in np.arange(10):
    axs[i,0].plot(range(n_timesteps), train_x[i*2,:,idx])
    axs[i,0].set_title("HDL changes for hospital stay i = {}".format(i*2))
    axs[i,1].plot(range(n_timesteps), train_x[i*2+1,:,idx])
    axs[i,1].set_title("HDL changes for hospital stay i = {}".format(i*2 + 1))
fig.tight_layout()

# END CODE #
```


HDL changes for hospital stay i = 0
HDL changes for hospital stay i = 1
HDL changes for hospital stay i = 2
HDL changes for hospital stay i = 3
HDL changes for hospital stay i = 4
HDL changes for hospital stay i = 5
HDL changes for hospital stay i = 6
HDL changes for hospital stay i = 7
HDL changes for hospital stay i = 8
HDL changes for hospital stay i = 9
HDL changes for hospital stay i = 10
HDL changes for hospital stay i = 11
HDL changes for hospital stay i = 12
HDL changes for hospital stay i = 13
HDL changes for hospital stay i = 14
HDL changes for hospital stay i = 15
HDL changes for hospital stay i = 16
HDL changes for hospital stay i = 17
HDL changes for hospital stay i = 18
HDL changes for hospital stay i = 19
1
0
0 2 4 6 8 10 12 14

By looking at these graphs and by carefully reading the description of `return_data` above, answer the following questions: - Why are the last values in the series usually 0, and why do some reach 0 earlier than others? - "HDL" is cholesterol, so how is it possible that some values are actually negative? - In many cases, the measured HDL cholesterol is exactly the same on consecutive days, even though we would expect a measurement to fluctuate at least a bit between days. Why is this the case?

1.1.3 Written answer:

- The last values are usually 0 due to zero padding. Since many patients are discharged at or prior to the 14 day period, we do not have HDL data for that individual for the remaining days. These missing entries are filled with 0 as described above. Some reach 0 earlier than others since some patients are discharged earlier than others.

- Since we Z-score normalize (normalize our values to mean = 0 and standard deviation = 1), it is possible to get negative normalized HDL values if the value is below the mean.
- This is likely due to inconsistent measurement between days. If we take a measurement at day1 and day5 but not in between, we interpolate day2, day3, and day4 to have the same value as day1. This could lead to identical HDL values on consecutive days.

Q1.6 prediction problem

In the first cell of this notebook we explained the prediction problem at a high level. Explain the prediction problem again, but be specific in explaining the data inputs and prediction outputs. Using the terminology from lecture, is this a "many-to-many" problem, or a "many-to-one" problem?

1.1.4 Written answer: In this notebook, we will be working with structured data measurements to make predictions of sepsis, myocardial infarction (MI), and vancomycin antibiotic administration. The data inputs are temporal dependent measurements, where each k = 226 features are measured at each 24 hour timestep block (for up to 15 timesteps) within a hospital stay. The input data is interpolated and padded where there's missing data, and the values are Z-score normalized. The output will be a binary prediction at each timestep for the onset of any of the above three conditions. This is a many-to-many problem since we have unique ICU data inputs at each timestep and are also making distinct predictions at each timestep.

1.2 LSTM prediction model

Q1.7 define an LSTM

We'll use an LSTM model to predict a patient outcome at each time step. So each data point we pass to the model will be a sequence of length `n_timesteps`, where each timestamp has `k` features. Our prediction output is also a sequence of length `n_timesteps`.

Using Keras `Sequential`, define a model called `model_lstm`. It should have: - 1 LSTM layer with 256 units. Use the default activation for the LSTM layer. Keras has optimized GPU implementations for most layers, but it does not have an optimized implementation for LSTM with non-default activations. - 1 dropout layer with `rate=0.5`. - 1 dense layer that applies the same tranformation to each of the prediction outputs. - A suitable activation function for the prediction task.

[50]:
```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Dense, Masking
```

[51]:
```python
def build_lstm_model(lstm_hidden_units=256):
    """
    Return a simple Keras model with a single LSTM layer, dropout later,
    and then dense prediction layer.

    Args:
    lstm_hidden_units (int): units in the LSTM layer

    Returns:
    model_lstm (tf.keras.Model) LSTM keras model with output dimension (None,1)
    """
    model_lstm = None
    # YOUR CODE HERE #
    model_lstm = Sequential()
    model_lstm.add(LSTM(lstm_hidden_units, return_sequences=True))
    model_lstm.add(Dropout(0.5))
    model_lstm.add(Dense(1, activation = 'sigmoid'))

    # END CODE #
    return model_lstm


# test code for checking the shape #
lstm_hidden_units = 256
model_lstm = build_lstm_model(lstm_hidden_units)
bs = 8
x_batch = train_x[:bs]
print(model_lstm(x_batch).shape)  # expect shape (8, 15, 1) # batch size 8, 15␣
 ↪timesteps
```

```
(8, 15, 1)
```

Q1.8 masking

Read about the masking layer in Keras. Briefly explain why we need masking for this problem. Your answer should refer back to the time-series plots generated in Q1.5.

1.2.1 Written answer: We will need to use masking to skip zero padded timesteps where the patient had missing data. In the time-series plots, we see that many patients plateau at zero before 14 days have gone by, indicating an event where data was no longer collected after a certain day (such as a discharge). These zero padded values should not be included in the model as they will affect prediction. Thus, we use masking to skip these timesteps.

The below function builds the final model. We provide code that calls `build_lstm_model`.

Your code should add a masking layer with `mask_value=0`, and it should be applied at the start of the model.

[52]:
```
def build_masked_lstm_model(num_timesteps, num_features, lstm_hidden_units=256):
    """
    Return a simple Keras model with a masking single LSTM layer, dropout later,
    and then dense prediction layer.

    Args:
    num_timesteps (int): num timesteps per input data object.
    num_features (int): num features per input data object.
    lstm_hidden_units (int): units in the LSTM layer

    Returns:
    model_lstm (tf.keras.Model) LSTM keras model with output dimension (None,1)
    """
    model_lstm = build_lstm_model(lstm_hidden_units)
    for layer in model_lstm.layers:
        layer.supports_masking = True

    model = None
    # YOUR CODE HERE #
    model = Sequential()
    model.add(tf.keras.layers.Masking(mask_value=0.,input_shape=(num_timesteps,␣
 ↪num_features)))
    model.add(model_lstm)
    # END CODE #
    return model


# Code to test the shape is correc #
num_timesteps, num_features = train_x.shape[-2:]
lstm_hidden_units = 256

model = build_masked_lstm_model(num_timesteps, num_features, lstm_hidden_units)
bs = 8
x_batch = train_x[:bs]
print(model(x_batch).shape)  # expect shape (8, 15, 1) # batch size 8, 15␣
 ↪timesteps
```

```
(8, 15, 1)
```

Q1.9 compiling and training

Below we have copied the code for getting the dataset that we ran earlier. You can choose ‘MI’‘, SEPSIS’ or ‘VANCOMYCIN’ as the target. When submitting this assignment, please choose ‘VANCOMYCIN’.

The code also calls `build_masked_lstm_model` that you just defined. Note that the model parameters depend on the dataset shape, so if we wanted to change the dataset from ‘MI’ to ‘SEPSIS’ then we need to create the model again with different input shapes. We chose to define the model inside a function so that we could re-create the model more easily.

Compile the model using - The Adam optimizer with default parameters. - An appropriate loss function for this task. - Metrics: accuracy and tensorflow’s AUC

[53]:
```
target = "VANCOMYCIN"  # 'SEPSIS' or 'MI' or 'VANCOMYCIN'
(
    train_x,
    val_x,
    train_y,
    val_y,
    no_feature_cols,
    test_x,
    test_y,
    x_boolmat_test,
    y_boolmat_test,
    x_boolmat_val,
    y_boolmat_val,
    features,
) = data_utils.load_seq_dataset(ROOT, target)
num_timesteps, num_features = train_x.shape[-2:]
model = build_masked_lstm_model(num_timesteps, num_features, lstm_hidden_units)

# YOUR CODE HERE #
optimizer = tf.keras.optimizers.Adam()
loss = tf.keras.losses.BinaryCrossentropy()
metrics = ['accuracy', tf.keras.metrics.AUC()]

model.compile(optimizer, loss, metrics)
# END CODE #
```

Finally fit the model. It should train very quickly. For ‘VANCOMYCIN’, validation accuracy should be around 0.85. For ‘MI’ it should be above 0.95.

[54]:
```
epochs = 10
batch_size = 16
# YOUR CODE HERE #
model.fit(
    train_x,
    train_y,
    batch_size = batch_size,
    epochs = epochs,
    validation_data = (val_x, val_y)
)

# END CODE #
```

```
Epoch 1/10
1814/1814 [==============================] - 19s 9ms/step - loss: 0.2377 -
accuracy: 0.8135 - auc_1: 0.8744 - val_loss: 0.1918 - val_accuracy: 0.8472 -
val_auc_1: 0.9045
Epoch 2/10
1814/1814 [==============================] - 14s 8ms/step - loss: 0.2133 -
accuracy: 0.8348 - auc_1: 0.9007 - val_loss: 0.1878 - val_accuracy: 0.8491 -
val_auc_1: 0.9084
Epoch 3/10
1814/1814 [==============================] - 15s 8ms/step - loss: 0.2050 -
accuracy: 0.8419 - auc_1: 0.9089 - val_loss: 0.1871 - val_accuracy: 0.8504 -
val_auc_1: 0.9085
Epoch 4/10
1814/1814 [==============================] - 14s 8ms/step - loss: 0.1992 -
accuracy: 0.8462 - auc_1: 0.9144 - val_loss: 0.1891 - val_accuracy: 0.8475 -
val_auc_1: 0.9066
Epoch 5/10
1814/1814 [==============================] - 14s 8ms/step - loss: 0.1934 -
accuracy: 0.8513 - auc_1: 0.9196 - val_loss: 0.1805 - val_accuracy: 0.8556 -
val_auc_1: 0.9179
Epoch 6/10
1814/1814 [==============================] - 14s 8ms/step - loss: 0.1888 -
accuracy: 0.8552 - auc_1: 0.9236 - val_loss: 0.1811 - val_accuracy: 0.8557 -
val_auc_1: 0.9176
Epoch 7/10
1814/1814 [==============================] - 15s 8ms/step - loss: 0.1839 -
accuracy: 0.8592 - auc_1: 0.9278 - val_loss: 0.1798 - val_accuracy: 0.8580 -
val_auc_1: 0.9169
Epoch 8/10
1814/1814 [==============================] - 15s 8ms/step - loss: 0.1782 -
accuracy: 0.8644 - auc_1: 0.9324 - val_loss: 0.1837 - val_accuracy: 0.8561 -
val_auc_1: 0.9158
Epoch 9/10
1814/1814 [==============================] - 15s 8ms/step - loss: 0.1729 -
accuracy: 0.8691 - auc_1: 0.9365 - val_loss: 0.1819 - val_accuracy: 0.8575 -
val_auc_1: 0.9156
Epoch 10/10
1814/1814 [==============================] - 15s 8ms/step - loss: 0.1662 -
accuracy: 0.8754 - auc_1: 0.9416 - val_loss: 0.1879 - val_accuracy: 0.8529 -
```

```
val_auc_1: 0.9130
```

[54]:
```
<keras.callbacks.History at 0x7f0a7de79b90>
```

1.2.2 Evaluation

Q1.10 predition and masking

Use `model.predict()` on the test dataset and save predictions to the variable `test_y_pred`.

[55]:
```
test_y_pred = None
# YOUR CODE HERE #
test_y_pred = model.predict(test_x)
# END CODE #

print(test_y_pred.shape)  # expect (n_datapoints, 15, 1)
```

```
(10355, 15, 1)
```

Normally when we we compare a set of predictions, we'd take 2 vectors: `y_true` which is a flat vector of 0s and 1s, and `y_pred` which is a vector of the same shape with probabilities in the range [0,1]. Our case is different because: - Our prediction output has an extra axis `(None,timesteps,features)` instead of `(None,features)`. - Some of the predictions should be masked, and therefore removed from the final prediction dataset.

The earlier function `data_utils.load_seq_dataset` returned a mask vector `y_boolmat_test` with the same shape as `test_y`. If `y_boolmat_test[i]==True` then this label should be masked (removed from the evaluation dataset).

In the next cell use `test_y_pred` and `y_boolmat_test` to create the vectors `y_pred_masked` and `y_true_masked` by removing the masked predictions, and flattening the output.

[56]:
```
y_pred_masked = None
y_true_masked = None

# YOUR CODE HERE #
y_pred_masked = np.array([])
y_true_masked = np.array([])

for i in np.arange(len(test_y)):
    zip_file_true = zip(test_y[i].flatten(), y_boolmat_test[i].flatten())
    zip_file_pred = zip(test_y_pred[i].flatten(), y_boolmat_test[i].flatten())
    y_true_masked = np.append(y_true_masked, [j[0] for j in zip_file_true if␣
↪j[1] == False])
    y_pred_masked = np.append(y_pred_masked, [j[0] for j in zip_file_pred if␣
↪j[1] == False])
# END CODE #
print(
    y_pred_masked.shape, y_true_masked.shape
)  # expect shape (n_predictions,) and the shape should be the same
```

```
(86947,) (86947,)
```

Q1.11 ROC+AUC

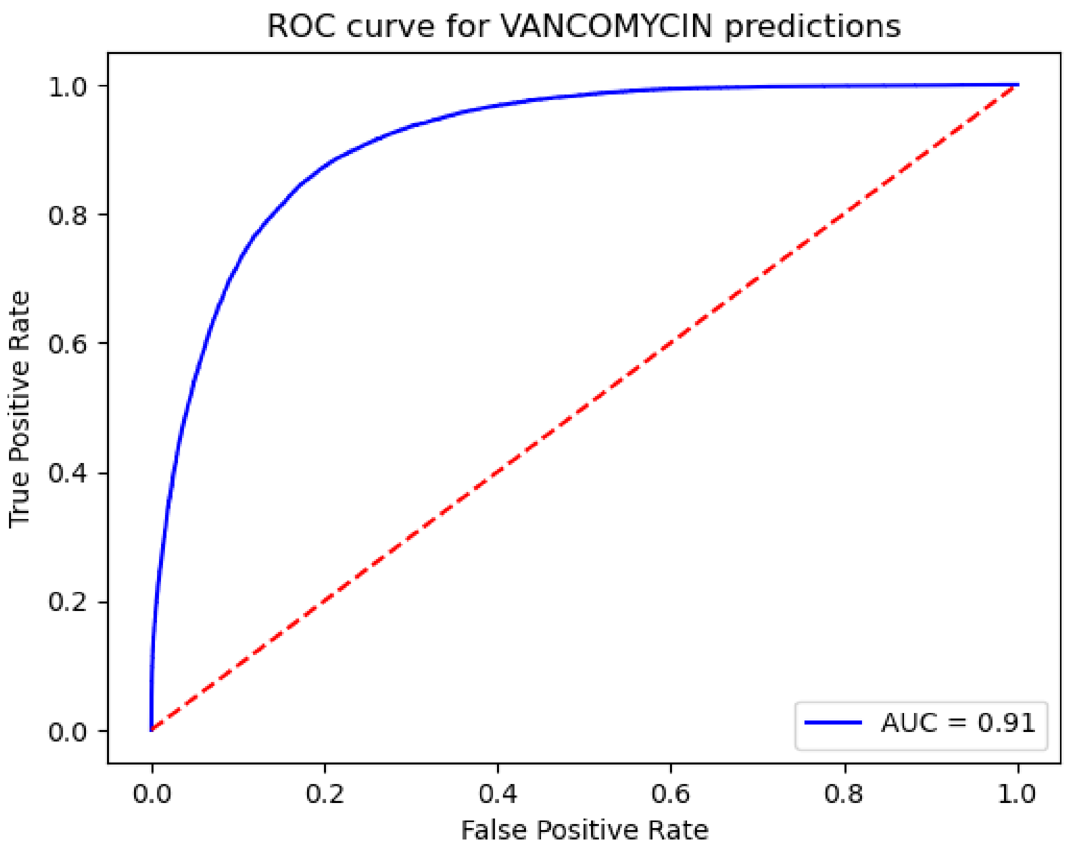
Using `y_pred_masked` and `y_true_masked`: - Plot a ROC curve. - Print the AUC.

You can use the functions in `sklearn.metrics` for both.

[59]:
```
import sklearn.metrics as metrics
```

[61]:
```
# YOUR CODE HERE #
fpr, tpr, threshold = metrics.roc_curve(y_true_masked, y_pred_masked)
auc = metrics.auc(fpr, tpr)
import matplotlib.pyplot as plt
plt.title('ROC curve for VANCOMYCIN predictions')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

# END CODE #
print(f"AUC is {auc}")
```



```
AUC is 0.9130219186375473
```

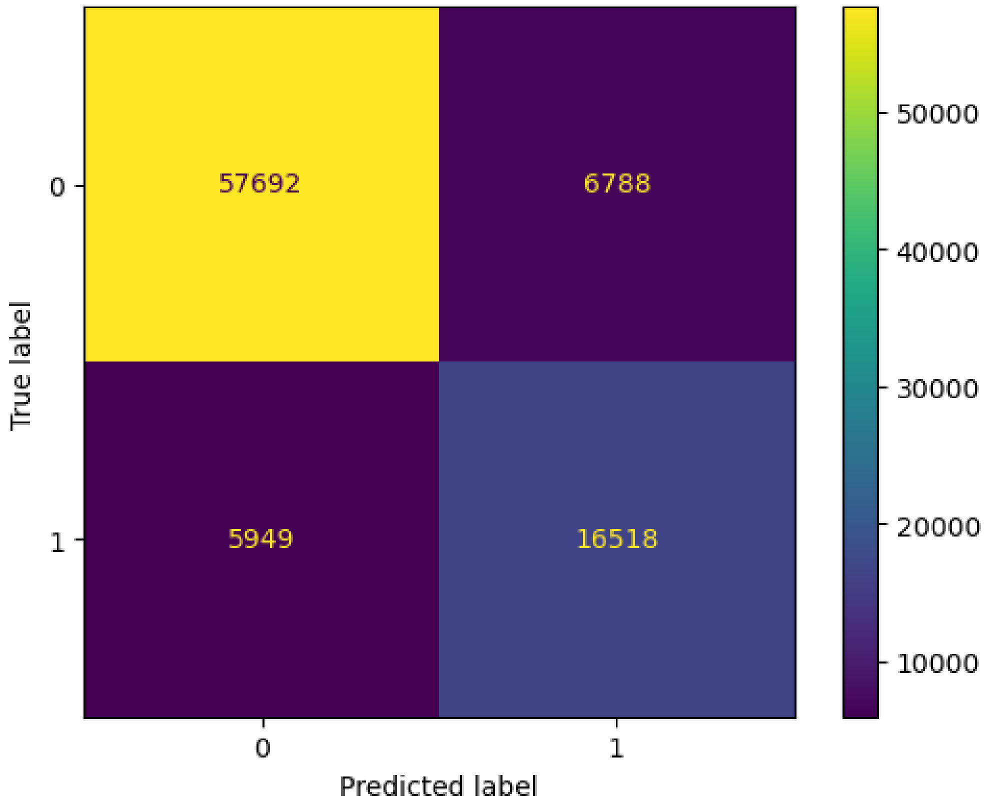
Q1.12 confusion

Finally, generate a confusion matrix. To do this you will need to convert prediction probabilities in `y_pred_masked` to binary predictions. You can choose a threshold of 0.5. Again, you can use functions from `sklearn.metrics`.

You can use a plotting library to display the confusion matrix, but you can also just print the array directly. If you do just print the array, then also print a message explaining the each axis.

[62]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

[63]:
```
# YOUR CODE HERE #
pred = y_pred_masked > 0.5
cf_matrix = confusion_matrix(y_true_masked, pred)
labels = [0, 1]
disp = ConfusionMatrixDisplay(confusion_matrix=cf_matrix, display_labels=labels)
disp.plot(include_values=True)
plt.show()
# END CODE #
```



Q1.13 clinical application tradeoffs

We have focused on modelling sepsis, but now consider vancomycin prediction. Explain what it means to choose different operating points at different positions in the ROC curve. Specifically tie it back to the use case of vancomycin antibiotic adminsitration. What are the tradeoffs? Pick a two points on the ROC curve and explain what the true positive and false positive rates mean at those points.

1.2.3 Written answer: The ROC curve displays performance of a classification model at all classification thresholds. Different operating points at different positions in the ROC curve represent the tradeoff between the true positive and false positive rates at a particular threshold. The best operating point (threshold) is chosen so that classifier achieves the optimal tradeoff between the cost of failing to detect positives and the costs of raising false alarms. For vancomycin antibiotic administration, an operating point in the ROC curve is a threshold value for which we can calculate the tradeoff between predicting necessary administration (true positives) and predicting the need for administration when the patient is healthy (false positives). If we would like to increase our probability that a patient in need of vancomycin administration is categorized correctly, we will likely tradeoff and get more false positives, healthy patients who are predicted as cases. Additionally, if we would like to decrease the probability that a healthy patient is predicted as needing vancomycin, we will likely attain more false negatives, patients in need who are predicted as healthy. A point on the ROC curve is approximately (0.092, 0.72) at threshold t = 0.52. This means that at this threshold, 9.2% of healthy patients were predicted as needing vancomycin and 72% of patients truly in need of vancomycin were predicted as such.

```
[ ]:
```

```
[ ]:
```