Github Link to Notebook and Files: https://github.com/ShivanVipani/CS598_Final

Note: Set runtime type to GPU to execute this entire notebook in Google Colab

Video Presentation Link: https://drive.google.com/file/d/10NjCOhyjdBFX__QF1AoOJ4-Pgh62tEis/view?usp=drivesdk

Data Link for Notebook Execution:

This is the data folder that I have loaded in the root of My Drive in google drive: https://drive.google.com/drive/folders/1HbImyO91P-3BAjdb_MGhQAAndeUMi7mX?usp=sharing

Mount Notebook to Google Drive

We are using files from the paper's git repo for preprocessing/training/evaluation, but have modified the files so they are executable within Google Colab. Those files will be loaded into this notebook through the above Drive folder, providing python scripts, executable sh scripts, and data from my experiments.

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

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

Introduction

Paper Introduction

- what did the paper propose: The paper aims to implement an ensemble learning approach for early diagnosis of sepsis using neural networks, specifically Recurrent Neural Networks (RNNs), trained on patient-specific data. It utilizes temporally-associated clinical measurements, demographic data, and ICU admission/discharge times to predict the onset of sepsis. Leveraging the ensemble's capability to combine diverse patient-specific models, the method seeks to enhance generalization and prediction accuracy.
- what is the innovations of the method: The method innovates by improving prediction accuracy through ensemble learning while addressing the challenges of privacy preservation in medical data.
- how well the proposed method work (in its own metrics): According to the paper's metrics, the proposed method achieved significant improvements in prediction accuracy compared to traditional models, particularly in terms of AUROC.
- what is the contribution to the research regime (referring the Background above, how important the paper is to the problem): Overall, the paper's contributions are significant as it offers a novel approach to sepsis prediction that balances accuracy and privacy concerns, addressing a critical need in the field of medical data analysis.

Background of the problem

- what type of problem: We are predicting sepsis, which occurs when chemicals released in the bloodstream to fight an infection trigger inflammation throughout the body.
- what is the importance/meaning of solving the problem: Sepsis can cause a cascade of changes that damage multiple organ systems, leading them to fail, sometimes even resulting in death, which is why it is so important to predict its onset and catch it as early as possible to prevent any damage to the body.
- what is the difficulty of the problem: Diagnosis is a difficult task when doctors have so much on their plates to crunch numbers on and look out for, and we have overwhelming amounts of population and personal patient data that is hard for the doctor to consume all at once.
- the state of the art methods and effectiveness:
    1. Machine Learning Techniques:

- These models leverage various patient data types, such as vital signs, laboratory results, and clinical notes, to identify complex patterns indicative of sepsis onset.
- Deep learning models like RNNs and CNNs, in particular, have shown promising results in capturing temporal dependencies in patient data and detecting subtle signs of sepsis.

2. Early Warning Systems (EWS):
   - EWSs continuously monitor patient data streams, including vital signs, laboratory values, and clinical observations, to detect early signs of deterioration, including sepsis.
   - These systems have been successful in triggering timely alerts to healthcare providers, enabling early intervention and improving patient outcomes.
   - EWSs are widely used in clinical settings to support proactive management of high-risk patients and reduce the incidence of sepsis-related complications.
3. Clinical Decision Support Systems (CDSS):
   - CDSSs integrate patient-specific data with clinical guidelines and evidence-based knowledge to assist healthcare providers in diagnosing and managing sepsis.
   - These systems provide real-time recommendations, alerts, and decision support tools to aid clinicians in identifying sepsis early and guiding appropriate treatment.
   - CDSSs have been shown to improve clinical decision-making, reduce diagnostic errors, and enhance patient safety in sepsis management.

Scope of Reproducibility:

The paper proposes that an ensemble approach to composing our model using patient-specific models would be more effective, and that creating an entire full model trained on the entirety of the data would yield a poor performing model. In my own time, I put in effort using Google Colab and its GPU engine to train those same models from the experiment, generating a full model and an ensemble model. However, there are approximately 17k patients, and with my hardware specifics available on colab, it would take approx 40 days to complete the experiment. So I had to assess my models on 1/8th of the amount of data by filtering the ids. I additionally created and trained models for my other experiments, involving data splitting and differing ensemble sizes, in this same fashion. Because the data is massive and there are so many patients to make patient-specific models for, I was not able to compete with the results of the authors, which contributed to lower accuracy outcomes than the paper as found below in this notebook. Though I was still impressed that it still did decently well.

For the purposes of reproducibility in this notebook, I've cut down the data to a subsection of patients to train on and build models, so that you can still see everything train and complete/be usable. The full model takes a very long time to train, so I have cut down its training to 10 epochs as well so that everything can run in this notebook in a timely manner while still displaying the functional execution of everything.

In the git repo, I will provide the files that were used for this adapted reproducible run, in addition to the files and notebooks that were used for my testing in my own time that I executed to get my results.

Execution time and a strong GPU available were definitely constraints for reproduction of this paper, as the paper needs a GPU and the only one I have available is the one provided by Google Colab.

Methodology

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

Environment

Python Version:

```
!python --version
```

```
Python 3.10.12
```

Dependencies/Packages Needed:

- PyTorch version >= 1.7.0
- numpy version >= 1.19.4
- scipy version >= 1.6.0
- Python version >= 3.6
- An NVIDIA GPU and NCCL

Data

Data Download:

```
!git clone https://github.com/statnlp/sepens
!cd sepens
!wget https://www.cl.uni-heidelberg.de/statnlpgroup/sepsisexp/SepsisExp.tar.gz
!tar zxvf SepsisExp.tar.gz
```

```
Cloning into 'sepens'...
remote: Enumerating objects: 53, done.
remote: Counting objects: 100% (53/53), done.
remote: Compressing objects: 100% (52/52), done.
remote: Total 53 (delta 21), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (53/53), 37.24 KiB | 3.10 MiB/s, done.
Resolving deltas: 100% (21/21), done.
--2024-05-07 21:08:16--  https://www.cl.uni-heidelberg.de/statnlpgroup/sepsisexp/SepsisExp.tar.gz
Resolving www.cl.uni-heidelberg.de (www.cl.uni-heidelberg.de)... 147.142.207.78
Connecting to www.cl.uni-heidelberg.de (www.cl.uni-heidelberg.de)|147.142.207.78|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 26373703 (25M) [application/x-gzip]
Saving to: 'SepsisExp.tar.gz'

SepsisExp.tar.gz    100%[===================>]  25.15M  87.4MB/s    in 0.3s

2024-05-07 21:08:16 (87.4 MB/s) - 'SepsisExp.tar.gz' saved [26373703/26373703]

README
sepsisexp_timeseries_partition-A.tsv
sepsisexp_timeseries_partition-B.tsv
sepsisexp_timeseries_partition-C.tsv
sepsisexp_timeseries_partition-D.tsv
```

```
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/make_data.sh" "/content/sepens/make_data.sh"
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/BI_make_data.sh" "/content/sepens/BI_make_data.sh"
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/TRI_make_data.sh" "/content/sepens/TRI_make_data.sh"

!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/SMALL_grow_ensemble_perrone.py" "/content/sepens/SMALL_gro
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/MEDIUM_grow_ensemble_perrone.py" "/content/sepens/MEDIUM_g
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/LARGE_grow_ensemble_perrone.py" "/content/sepens/LARGE_gro

!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/make_fullmodel.sh" "/content/sepens/make_fullmodel.sh"
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/make_models_perpat.sh" "/content/sepens/make_models_perpat
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/make_fullmodel.sh" "/content/sepens/make_fullmodel.sh"

!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/inference_fullmodel.sh" "/content/sepens/inference_fullmod
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/inference_poolmodels.sh" "/content/sepens/inference_poolmo
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/inference_ensmodels.sh" "/content/sepens/inference_ensmode

!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/calc_auroc.py" "/content/sepens/calc_auroc.py"
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/dev_ids.py" "/content/sepens/dev_ids.py"
```

```
!cp "/content/drive/My Drive/DL4H_Final_Files/main_root/dev_ids.py" "/content/dev_ids.py"
!cp "/content/drive/My Drive/DL4H_Final_Files/main_root/sorted_filter_train_ids.dat" "/content/sorted_filter_train_ids
!cp "/content/drive/My Drive/DL4H_Final_Files/main_root/test_ids.dat" "/content/test_ids.dat"

!cp "/content/drive/My Drive/DL4H_Final_Files/data/data_2way/sepsis_partition-E.tsv" "/content/sepsis_partition-E.tsv"
!cp "/content/drive/My Drive/DL4H_Final_Files/data/data_2way/sepsis_partition-F.tsv" "/content/sepsis_partition-F.tsv"
!cp "/content/drive/My Drive/DL4H_Final_Files/data/data_3way/sepsis_partition-G.tsv" "/content/sepsis_partition-G.tsv"
!cp "/content/drive/My Drive/DL4H_Final_Files/data/data_3way/sepsis_partition-H.tsv" "/content/sepsis_partition-H.tsv"
!cp "/content/drive/My Drive/DL4H_Final_Files/data/data_3way/sepsis_partition-I.tsv" "/content/sepsis_partition-I.tsv"
!cp "/content/drive/My Drive/DL4H_Final_Files/data/data_4way/sepsisexp_timeseries_partition-A.tsv" "/content/sepsisexp
!cp "/content/drive/My Drive/DL4H_Final_Files/data/data_4way/sepsisexp_timeseries_partition-B.tsv" "/content/sepsisexp
!cp "/content/drive/My Drive/DL4H_Final_Files/data/data_4way/sepsisexp_timeseries_partition-C.tsv" "/content/sepsisexp
!cp "/content/drive/My Drive/DL4H_Final_Files/data/data_4way/sepsisexp_timeseries_partition-D.tsv" "/content/sepsisexp

!cp "/content/drive/My Drive/DL4H_Final_Files/results.png" "/content/results.png"
```

Data Visualizations and Statistics:

The first split file from each split experiment group is visualized for its class count and its statistic below. The rest of its related split files in their respective groups look similar, so I thought to give a sample from each experiment group rather than clustering the notebook with the same graphs.

```
# Original 4-Way Split
dfA = pd.read_csv('/content/sepsisexp_timeseries_partition-A.tsv', sep='\t')
#dfB = pd.read_csv('/content/sepsisexp_timeseries_partition-B.tsv', sep='\t')
#dfC = pd.read_csv('/content/sepsisexp_timeseries_partition-C.tsv', sep='\t')
#dfD = pd.read_csv('/content/sepsisexp_timeseries_partition-D.tsv', sep='\t')

# 2-Way Split
dfE = pd.read_csv('/content/sepsis_partition-E.tsv', sep='\t')
#dfF = pd.read_csv('/content/sepsis_partition-F.tsv', sep='\t')

# 3-Way Split
dfG = pd.read_csv('/content/sepsis_partition-G.tsv', sep='\t')
#dfH = pd.read_csv('/content/sepsis_partition-H.tsv', sep='\t')
#dfI = pd.read_csv('/content/sepsis_partition-I.tsv', sep='\t')
```

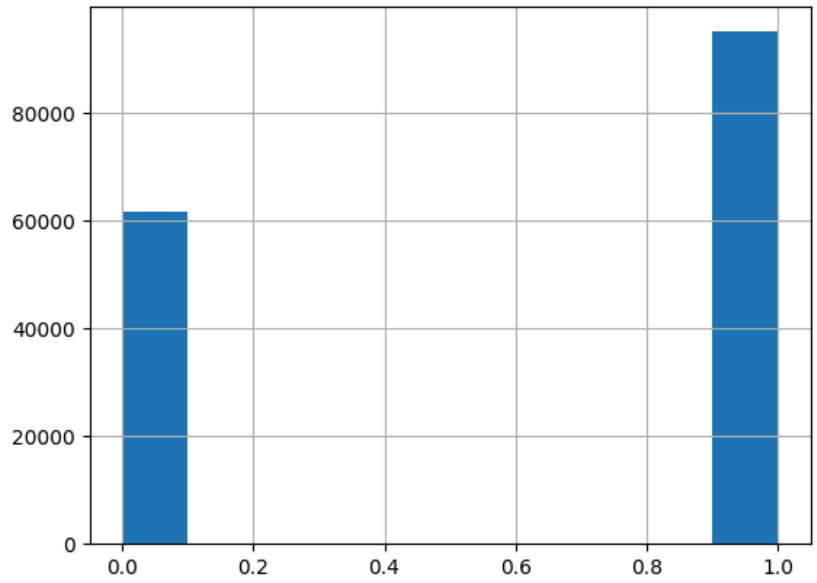
```
dfA.describe()
```

	id	sepsis	severity	timestep	respiratory_minute_volume	heart_rate	leukocytes
count	156704.000000	156704.000000	156704.000000	156704.000000	156704.000000	156704.000000	156704.000000
mean	12626.643883	0.606692	1.198712	415.394856	-0.158946	-0.109885	-0.063523
std	716.262248	0.488486	1.445370	483.193528	1.116563	0.984716	0.841941
min	11460.000000	0.000000	0.000000	0.000000	-3.033777	-4.288378	-1.777270
25%	12066.000000	0.000000	0.000000	83.500000	-0.606959	-0.887358	-0.622171
50%	12453.000000	1.000000	1.000000	232.000000	-0.241275	-0.207154	-0.219299
75%	13201.000000	1.000000	3.000000	565.500000	0.390363	0.521636	0.338181
max	14121.000000	1.000000	4.000000	2681.000000	3.615038	4.942962	6.003663

8 rows × 47 columns

```
dfA['sepsis'].hist()
```

```
<Axes: >
```



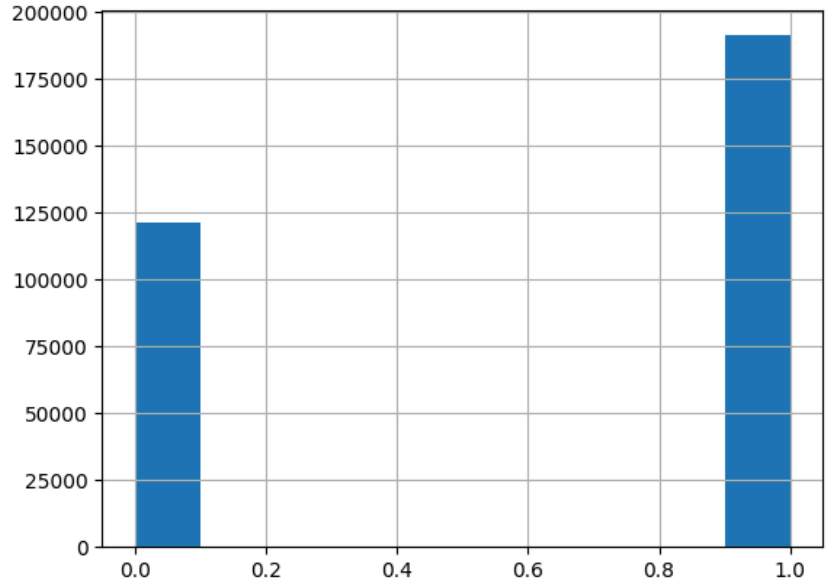

```
dfE.describe()
```

	id	sepsis	severity	timestep	respiratory_minute_volume	heart_rate	leukocytes
count	312923.000000	312923.000000	312923.000000	312923.000000	312923.000000	312923.000000	312923.000000
mean	12683.764396	0.611368	1.178405	398.792439	-0.143812	-0.130647	-0.100515
std	714.633166	0.487440	1.430222	453.685705	1.127586	0.966838	0.866519
min	11460.000000	0.000000	0.000000	0.000000	-3.033777	-4.288378	-1.856060
25%	12122.000000	0.000000	0.000000	83.000000	-0.640204	-0.838772	-0.697989
50%	12599.000000	1.000000	1.000000	231.000000	-0.241275	-0.207154	-0.256465
75%	13254.000000	1.000000	2.000000	541.500000	0.456851	0.473050	0.289122
max	14121.000000	1.000000	4.000000	2681.000000	3.615038	5.477408	6.772242

8 rows × 47 columns

```
dfE['sepsis'].hist()
```

```
<Axes: >
```



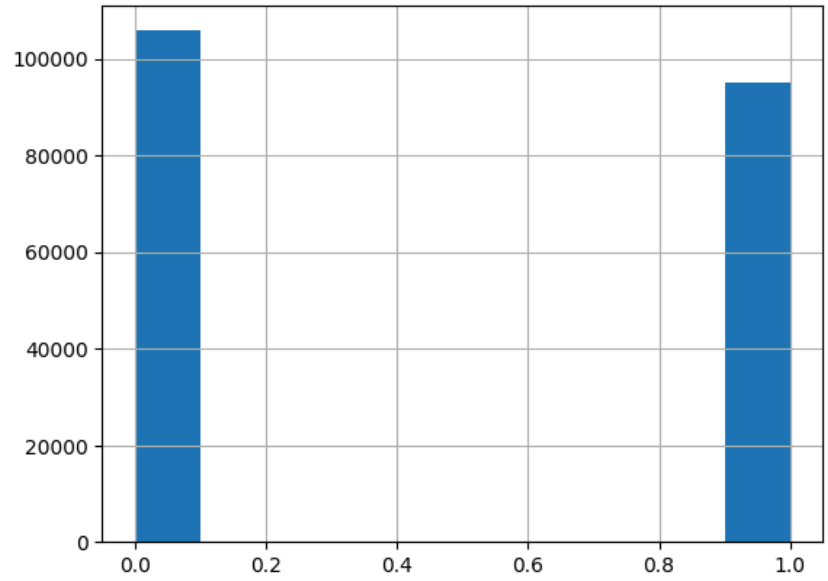
```
dfG.describe()
```

	id	sepsis	severity	timestep	respiratory_minute_volume	heart_rate	leukocytes
count	200856.000000	200856.000000	200856.000000	200856.000000	200856.000000	200856.000000	200856.000000
mean	12706.094685	0.473329	0.997755	360.511105	-0.195945	-0.172993	-0.084188
std	719.542599	0.499289	1.348313	448.417271	1.080982	0.967198	0.818716
min	11460.000000	0.000000	0.000000	0.000000	-3.033777	-4.288378	-1.856060
25%	12128.000000	0.000000	0.000000	66.500000	-0.640204	-0.887358	-0.635551
50%	12594.000000	0.000000	0.000000	190.000000	-0.307763	-0.304326	-0.243085
75%	13376.000000	1.000000	1.000000	471.000000	0.357119	0.424464	0.302502
max	14121.000000	1.000000	4.000000	2681.000000	3.615038	5.137306	6.003663

8 rows × 47 columns

```
dfG['sepsis'].hist()
```

```
<Axes: >
```



Training/Val Splits:

- Original Dataset Split (4-way): ~ 70/30
- 2-way Split: 50/50
- 3-way Split: ~ 67/33

Preprocessing Code:

```
!chmod +x sepens/make_data.sh
!./sepens/make_data.sh 0
```

```
INFO: Data files exist, generating train/dev/test splits...
12290 12296 12298 12334 12342 12349 12350 12351 13914 12361 12364 12376 12377 12383 12415 12426 12429 12434 12438
12299 11795 12821 11804 13915 13348 12844 13662 12346 11838 13377 11843 13324 12880 13417 12909 12922 13933 12930
Written to dev_ids_s0.dat
12290,12296,12298,12334,12342,12349,12350,12351,13914,12361,12364,12376,12377,12383,12415,12426,12429,12434,12438
12299,11795,12821,11804,13915,13348,12844,13662,12346,11838,13377,11843,13324,12880,13417,12909,12922,13933,12930
Written to ./dev_ids_s0.inc and dev_ids.py.
12315 12318 12348 14009 12381 12384 12397 12402 12403 12421 12435 12446 12451 12452 12455 12456 12457 12459 12463
12805 12294 12974 12812 13328 13144 12678 13858 11814 11819 11826 12855 11861 13403 11873 12387 13926 11880 12394
Written to test_ids_s0.dat
```

```
12315,12318,12348,14009,12381,12384,12397,12402,12403,12421,12435,12446,12451,12452,12455,12456,12457,12459,12463
12805,12294,12974,12812,13328,13144,12678,13858,11814,11819,11826,12855,11861,13403,11873,12387,13926,11880,12394
Written to ./test_ids_s0.inc and test_ids.py
```

For running the bi-split or tri-split versions of my own experiments (the original execution is a 4-way split) for pre-processing and generating the data that the model uses, you can uncomment your choice of the two cells below and run those instead of the cell above.

```
#!chmod +x sepens/BI_make_data.sh
#!./sepens/BI_make_data.sh 0
```

```
#!chmod +x sepens/TRI_make_data.sh
#!./sepens/TRI_make_data.sh 0
```

Model

Citation Reference of Original Paper:

1. Schamoni, S., Hagmann, M., & Riezler, S. (2022). Ensembling Neural Networks for Improved Prediction and Privacy in Early Diagnosis of Sepsis. Proceedings of the 7th Machine Learning for Healthcare Conference, PMLR 182, 123-145. Retrieved from https://doi.org/10.48550/arXiv.2209.00439

```
@inproceedings{schamoni2022,
  author = {Schamoni, Shigehiko and Hagmann, Michael and Riezler, Stefan},
  title = {Ensembling Neural Networks for Improved Prediction and Privacy in Early Diagnosis of Sepsis},
  booktitle = {Proceedings of the 6th Machine Learning for Healthcare Conference},
  year = {2022},
  city = {Durham, NC},
  country = {USA},
  volume = {182},
  series = {Proceedings of Machine Learning Research},
  publisher = {PMLR},
  url = {https://www.cl.uni-heidelberg.de/~schamoni/publications/dl/MLHC2022_Ensembling.pdf}
}
```

Link to the Original Paper's Repo:

https://github.com/StatNLP/sepens/tree/main

Model Descriptions:

Model Architecture:

It is defined in model_ts.py, a module provided by the paper's repo, that will be visualized below for viewing. The model is not pre-trained, it is trained from scratch off the data we load. The model is trained and validated in main_ts.py, which will be visualized below as well.

2 layers, 200 hidden units per layer

Input Layer:

- Linear layer mapping input features to a specified size.

Recurrent Neural Network (RNN) Layer:

- LSTM: Long-Short Term Memory model.
- Applies dropout for regularization.

Fully Connected (FC) Layer:

- Linear layer mapping RNN output to a single output neuron.

Dropout Layer:

- Applied before input layer and after RNN layer.

Weight Initialization:

- Weights are initialized uniformly, biases are initialized to zero.

Forward Pass:

- Input passes through layers sequentially, reshaping before FC layer.

Hidden State Initialization:

- Initial hidden state is set based on the RNN type.

Training Objectives:

Loss: MSELoss()

Optimizer: torch.nn.utils.clip_grad_norm_()

Implementation Code:

model_ts.py:

```
import torch.nn as nn

class RNNModelTS(nn.Module):
    """Container module with an encoder (inp), a recurrent module, and a decoder (fc)."""

    def __init__(self, rnn_type, nfeat, ninp, nhid, nlayers, dropout=0.5):
        super(RNNModelTS, self).__init__()
        self.drop = nn.Dropout(dropout)
        self.inp = nn.Linear(nfeat, ninp)
        if rnn_type in ['LSTM', 'GRU']:
            self.rnn = getattr(nn, rnn_type)(ninp, nhid, nlayers, dropout=dropout)
        else:
            try:
                nonlinearity = {'RNN_TANH': 'tanh', 'RNN_RELU': 'relu'}[rnn_type]
            except KeyError:
                raise ValueError( """An invalid option for `--model` was supplied,
                                  options are ['LSTM', 'GRU', 'RNN_TANH' or 'RNN_RELU']""")
            self.rnn = nn.RNN(ninp, nhid, nlayers, nonlinearity=nonlinearity, dropout=dropout)
        self.fc = nn.Linear(nhid, 1)

        self.rnn_type = rnn_type
        self.nhid = nhid
        self.nlayers = nlayers

    def init_weights(self):
        initrange = 0.1
        self.inp.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()
        self.fc.weight.data.uniform_(-initrange, initrange)

    def forward(self, input, hidden):

        emb = self.drop(self.inp(input))
        output, hidden = self.rnn(emb, hidden)
```

```
        output = self.drop(output)
        #
        decoded = self.fc(output.view(output.size(0)*output.size(1), output.size(2)))
        return decoded.view(output.size(0), output.size(1), decoded.size(1)), hidden

    def init_hidden(self, bsz):
        weight = next(self.parameters())
        if self.rnn_type == 'LSTM':
            return (weight.new_zeros(self.nlayers, bsz, self.nhid),
                    weight.new_zeros(self.nlayers, bsz, self.nhid))
        else:
            return weight.new_zeros(self.nlayers, bsz, self.nhid)
```

main_ts.py:

```
import argparse
import time
import math
import os
import torch
import torch.nn as nn
import torch.onnx
import data
import model_ts


parser = argparse.ArgumentParser(description='PyTorch SCIDATOS Time Series RNN/LSTM Model')
parser.add_argument('--data', type=str, default='.',
                    help='location of the data corpus')
parser.add_argument('--model', type=str, default='LSTM',
                    help='type of recurrent net (RNN_TANH, RNN_RELU, LSTM, GRU)')
parser.add_argument('--nfeatures', type=int, default=43,
                    help='number of input features (time series)')
parser.add_argument('--insize', type=int, default=200,
                    help='size of input for RNN')
parser.add_argument('--nhid', type=int, default=200,
                    help='number of hidden units per layer')
parser.add_argument('--nlayers', type=int, default=2,
                    help='number of layers')
parser.add_argument('--lr', type=float, default=20,
                    help='initial learning rate')
parser.add_argument('--clip', type=float, default=0.25,
                    help='gradient clipping')
parser.add_argument('--epochs', type=int, default=40,
                    help='upper epoch limit')
parser.add_argument('--min_epochs', type=int, default=10,
                    help='minimum epoch before early stopping')
parser.add_argument('--batch_size', type=int, default=20, metavar='N',
                    help='batch size')
parser.add_argument('--bptt', type=int, default=48,
                    help='sequence length')
parser.add_argument('--seqoverlap', type=float, default=0.5,
                    help='sequence overlap')
parser.add_argument('--dropout', type=float, default=0.2,
                    help='dropout applied to layers (0 = no dropout)')
parser.add_argument('--tied', action='store_true',
                    help='tie the word embedding and softmax weights')
parser.add_argument('--seed', type=int, default=1111,
                    help='random seed')
```

```
parser.add_argument('--cuda', action='store_true',
                    help='use CUDA')
parser.add_argument('--log-interval', type=int, default=100, metavar='N',
                    help='report interval')
parser.add_argument('--save', type=str, default='model.pt',
                    help='path to save the final model')
parser.add_argument('--onnx-export', type=str, default='',
                    help='path to export the final model in onnx format')
args = parser.parse_args()

# Set the random seed manually for reproducibility.
torch.manual_seed(args.seed)
if torch.cuda.is_available():
    if not args.cuda:
        print("WARNING: You have a CUDA device, so you should probably run with --cuda")

device = torch.device("cuda" if args.cuda else "cpu")

###############################################################################
# Load data
###############################################################################

# Load timeseries as torch tensor (cannot be modified)
# timeseries = data.TimeseriesTorch(args.data, args.nfeatures)

# Load timeseries as numpy array (not immutable)
timeseries = data.TimeseriesNumPy(args.data, args.nfeatures)
print("Timeseries loaded")

# Starting from sequential data, batchify arranges the dataset into columns.
# For instance, with the alphabet as the sequence and batch size 4, we'd get
# ┌ a g m s ┐
# │ b h n t │
# │ c i o u │
# │ d j p v │
# │ e k q w │
# └ f l r x ┘.
# These columns are treated as independent by the model, which means that the
# dependence of e. g. 'g' on 'f' can not be learned, but allows more efficient
# batch processing.

def batchify(rawdata, bsz):
    # Work out how cleanly we can divide the dataset into bsz parts.
    nbatch = rawdata[0].size(0) // bsz
    # Trim off any extra elements that wouldn't cleanly fit (remainders).
    data = rawdata[0].narrow(0, 0, nbatch * bsz)

    # Evenly divide the data across the bsz batches.
    data = data.view(-1, nbatch, args.nfeatures)
    data = data.permute(1,0,2).contiguous()

    ys = rawdata[1].narrow(0, 0, nbatch * bsz)
    ys = ys.view(-1, nbatch, 1)
    ys = ys.permute(1,0,2).contiguous()

    return [data.to(device),ys.to(device)]



# Generate overlapping sequences to avoid information loss by unfavorable
```

```
# sequence splits

def expand_and_batchify(rawdata, bsz, step=0.5):

    seqlen = args.bptt
    rawlen = int(len(rawdata[0]))

    stepsize = int(seqlen*step)

    # calculate length of new sequence with overlapa
    timesteps = (math.floor((rawlen-seqlen)/stepsize)+1 ) * seqlen
    timesteps += rawlen - (math.floor((rawlen-seqlen)/stepsize)+1) * stepsize

    steps = torch.Tensor(timesteps, args.nfeatures)
    targets = torch.Tensor(timesteps)    # severity

    pos = 0
    for i in range(0, rawlen-seqlen+1, stepsize):
        for j in range(0,seqlen):
            steps[pos] = torch.from_numpy(rawdata[0][i + j])
            targets[pos] = float(rawdata[1][i + j])
            pos += 1
    # copy over remainders
    remainderstart = (math.floor((rawlen-seqlen)/stepsize)+1) * stepsize
    for k in range(remainderstart, rawlen):
        steps[pos] = torch.from_numpy(rawdata[0][k])
        targets[pos] = float(rawdata[1][k])
        pos += 1

    # make sure we have initialized every position
    assert(timesteps == pos)

    return batchify([steps,targets], bsz)




eval_batch_size = 10

train_data = expand_and_batchify(timeseries.train, args.batch_size)
val_data = expand_and_batchify(timeseries.valid, eval_batch_size)
test_data = expand_and_batchify(timeseries.test, eval_batch_size)


###############################################################################
# Build the model
###############################################################################

model = model_ts.RNNModelTS(args.model, args.nfeatures, args.insize,
                            args.nhid, args.nlayers, args.dropout).to(device)

criterion = nn.MSELoss()

###############################################################################
# Training code
###############################################################################

def repackage_hidden(h):
    """Wraps hidden states in new Tensors, to detach them from their history."""
    if isinstance(h, torch.Tensor):
```

```
            return h.detach()
        else:
            return tuple(repackage_hidden(v) for v in h)


    # get_batch subdivides the source data into chunks of length args.bptt.
    # If source is equal to the example output of the batchify function, with
    # a bptt-limit of 2, we'd get the following two Variables for i = 0:
    # ┌ a g m s ┐ ┌ b h n t ┐
    # └ b h n t ┘ └ c i o u ┘
    # Note that despite the name of the function, the subdivison of data is not
    # done along the batch dimension (i.e. dimension 1), since that was handled
    # by the batchify function. The chunks are along dimension 0, corresponding
    # to the seq_len dimension in the LSTM.

    def get_batch(source, i):

        seq_len = min(args.bptt, len(source[0]) - 1 - i)
        data = source[0][i:i+seq_len]

        target = source[1][i:i+seq_len]

        return data, target

    def flatten(mydata):
        return mydata.permute(1,0,2).contiguous().view(-1,1)

    def evaluate(data_source):
        # Turn on evaluation mode which disables dropout.
        model.eval()
        total_loss = 0.

        hidden = model.init_hidden(eval_batch_size)
        with torch.no_grad():
            for i in range(0, data_source[0].size(0) - 1, args.bptt):

                data, targets = get_batch(data_source, i)
                output, hidden = model(data, hidden)

                total_loss += len(data) * criterion(output, targets).item()
                hidden = repackage_hidden(hidden)

        return total_loss / (len(data_source[0]) - 1)


    def train():
        # Turn on training mode which enables dropout.
        model.train()
        total_loss = 0.
        total_xentropy = 0.
        start_time = time.time()

        hidden = model.init_hidden(args.batch_size)
        for batch, i in enumerate(range(0, train_data[0].size(0) - 1, args.bptt)):
            data, targets = get_batch(train_data, i)

            hidden = repackage_hidden(hidden)
            model.zero_grad()
            output, hidden = model(data, hidden)
```

```
            loss = criterion(output, targets)

            loss.backward()

            # `clip_grad_norm` helps prevent the exploding gradient problem in RNNs / LSTMs.
            torch.nn.utils.clip_grad_norm_(model.parameters(), args.clip)
            for p in model.parameters():
                p.data.add_(-lr, p.grad.data)

            total_loss += loss.item()

            if batch % args.log_interval == 0 and batch > 0:
                cur_loss = total_loss / args.log_interval
                cur_xentropy = total_xentropy / args.log_interval
                elapsed = time.time() - start_time
                print('| epoch {:3d} | {:5d}/{:5d} batches | lr {:02.2f} | ms/batch {:5.2f} | '
                        'loss {:5.2f} | mse {:8.5f}'.format(
                    epoch, batch, len(train_data[0]) // args.bptt, lr,
                    elapsed * 1000 / args.log_interval, cur_loss, cur_loss))
                total_loss = 0
                start_time = time.time()


    def export_onnx(path, batch_size, seq_len):
        print('The model is also exported in ONNX format at {}'.
              format(os.path.realpath(args.onnx_export)))
        model.eval()
        dummy_input = torch.LongTensor(seq_len * batch_size).zero_().view(-1, batch_size).to(device)
        hidden = model.init_hidden(batch_size)
        torch.onnx.export(model, (dummy_input, hidden), path)


    # Loop over epochs.
    lr = args.lr
    best_val_loss = None

    # At any point you can hit Ctrl + C to break out of training early.
    try:
        for epoch in range(1, args.epochs+1):
            epoch_start_time = time.time()
            train()
            val_loss = evaluate(val_data)
            print('-' * 89)
            print('| end of epoch {:3d} | time: {:5.2f}s | valid loss {:5.2f} | '
                    'valid mse {:8.5f}'.format(epoch, (time.time() - epoch_start_time),
                                               val_loss, val_loss))
            print('-' * 89)
            # Save the model if the validation loss is the best we've seen so far.
            if epoch >= args.min_epochs and (not best_val_loss or val_loss < best_val_loss):
                with open(args.save, 'wb') as f:
                    torch.save(model, f)
                best_val_loss = val_loss
            else:
                # Anneal the learning rate if no improvement has been seen in the validation dataset.
                lr /= 4.0

    except KeyboardInterrupt:
        print('-' * 89)
        print('Exiting from training early')
```

```
# Load the best saved model.
with open(args.save, 'rb') as f:
    model = torch.load(f)
    # after load the rnn params are not a continuous chunk of memory
    # this makes them a continuous chunk, and will speed up forward pass
    model.rnn.flatten_parameters()

# Run on test data.
test_loss = evaluate(test_data)
print('=' * 89)
print('| End of training | test loss {:5.2f} | test mse {:8.5f}'.format(
    test_loss, test_loss))
print('=' * 89)

if len(args.onnx_export) > 0:
    # Export the model in ONNX format.
    export_onnx(args.onnx_export, batch_size=1, seq_len=args.bptt)
```

I have saved some pretrained models in the git repo, but we will be working with live trained models in this notebook.

Training

Hyperparams:

- Learning Rate: 20
- Batch Size: 20
- Dropout: 0.2

Computational Requirements:

- Num of Training Epochs:
  - My Own Training
    - Full Model: 200
    - Patient-Specific Models: 20
  - This Notebook
    - Full Model: 10
    - Patient-Specific Models: 20
- Hardware Requirements: Nvidia GPU
- Average Runtime for Each Epoch:
  - My Own Training
    - Full Model: ~ 40 sec
    - Patient-Specific Models: ~ 10 sec
  - This Notebook
    - Full Model: ~ 5 sec
    - Patient-Specific Models: ~ 1 sec
- GPU Hrs Used: ~ 5 Days

Training Code:

```
!chmod +x sepens/make_fullmodel.sh
```

```
!./sepens/make_fullmodel.sh
```

```
-----------------------------------------------------------------------------------------
| epoch   5 |   100/  643 batches | lr 0.08 | ms/batch  6.52 | loss  0.59 | mse  0.59046
| epoch   5 |   200/  643 batches | lr 0.08 | ms/batch  7.26 | loss  0.68 | mse  0.68034
| epoch   5 |   300/  643 batches | lr 0.08 | ms/batch  7.00 | loss  0.57 | mse  0.57250
| epoch   5 |   400/  643 batches | lr 0.08 | ms/batch  7.28 | loss  0.45 | mse  0.44998
| epoch   5 |   500/  643 batches | lr 0.08 | ms/batch  7.24 | loss  0.37 | mse  0.37087
| epoch   5 |   600/  643 batches | lr 0.08 | ms/batch  7.07 | loss  0.46 | mse  0.46239
-----------------------------------------------------------------------------------------
| end of epoch   5 | time:  6.09s | valid loss  2.22 | valid mse  2.21813
-----------------------------------------------------------------------------------------
| epoch   6 |   100/  643 batches | lr 0.02 | ms/batch  7.40 | loss  0.58 | mse  0.57946
| epoch   6 |   200/  643 batches | lr 0.02 | ms/batch  6.97 | loss  0.71 | mse  0.70558
| epoch   6 |   300/  643 batches | lr 0.02 | ms/batch  6.97 | loss  0.56 | mse  0.55765
| epoch   6 |   400/  643 batches | lr 0.02 | ms/batch  5.11 | loss  0.45 | mse  0.44688
| epoch   6 |   500/  643 batches | lr 0.02 | ms/batch  4.55 | loss  0.37 | mse  0.37472
| epoch   6 |   600/  643 batches | lr 0.02 | ms/batch  4.65 | loss  0.46 | mse  0.45779
-----------------------------------------------------------------------------------------
| end of epoch   6 | time:  4.75s | valid loss  1.87 | valid mse  1.86768
-----------------------------------------------------------------------------------------
| epoch   7 |   100/  643 batches | lr 0.00 | ms/batch  4.79 | loss  0.52 | mse  0.51798
| epoch   7 |   200/  643 batches | lr 0.00 | ms/batch  4.82 | loss  0.66 | mse  0.65642
| epoch   7 |   300/  643 batches | lr 0.00 | ms/batch  4.76 | loss  0.52 | mse  0.52410
| epoch   7 |   400/  643 batches | lr 0.00 | ms/batch  4.88 | loss  0.43 | mse  0.43344
| epoch   7 |   500/  643 batches | lr 0.00 | ms/batch  4.70 | loss  0.35 | mse  0.34954
| epoch   7 |   600/  643 batches | lr 0.00 | ms/batch  4.76 | loss  0.46 | mse  0.46106
-----------------------------------------------------------------------------------------
| end of epoch   7 | time:  4.05s | valid loss  1.91 | valid mse  1.90516
-----------------------------------------------------------------------------------------
| epoch   8 |   100/  643 batches | lr 0.00 | ms/batch  4.77 | loss  0.52 | mse  0.52383
| epoch   8 |   200/  643 batches | lr 0.00 | ms/batch  4.79 | loss  0.65 | mse  0.65117
| epoch   8 |   300/  643 batches | lr 0.00 | ms/batch  4.58 | loss  0.51 | mse  0.51040
| epoch   8 |   400/  643 batches | lr 0.00 | ms/batch  5.32 | loss  0.44 | mse  0.44349
| epoch   8 |   500/  643 batches | lr 0.00 | ms/batch  4.62 | loss  0.36 | mse  0.35566
| epoch   8 |   600/  643 batches | lr 0.00 | ms/batch  4.71 | loss  0.47 | mse  0.46553
-----------------------------------------------------------------------------------------
| end of epoch   8 | time:  4.31s | valid loss  1.92 | valid mse  1.91781
-----------------------------------------------------------------------------------------
| epoch   9 |   100/  643 batches | lr 0.00 | ms/batch  6.30 | loss  0.52 | mse  0.52408
| epoch   9 |   200/  643 batches | lr 0.00 | ms/batch  6.57 | loss  0.65 | mse  0.65032
| epoch   9 |   300/  643 batches | lr 0.00 | ms/batch  6.77 | loss  0.51 | mse  0.51346
| epoch   9 |   400/  643 batches | lr 0.00 | ms/batch  7.15 | loss  0.44 | mse  0.44418
| epoch   9 |   500/  643 batches | lr 0.00 | ms/batch  6.80 | loss  0.36 | mse  0.35887
| epoch   9 |   600/  643 batches | lr 0.00 | ms/batch  6.99 | loss  0.47 | mse  0.47203
-----------------------------------------------------------------------------------------
| end of epoch   9 | time:  5.87s | valid loss  1.92 | valid mse  1.92063
-----------------------------------------------------------------------------------------
| epoch  10 |   100/  643 batches | lr 0.00 | ms/batch  7.13 | loss  0.52 | mse  0.51780
| epoch  10 |   200/  643 batches | lr 0.00 | ms/batch  7.02 | loss  0.65 | mse  0.64998
| epoch  10 |   300/  643 batches | lr 0.00 | ms/batch  7.18 | loss  0.52 | mse  0.51739
| epoch  10 |   400/  643 batches | lr 0.00 | ms/batch  7.02 | loss  0.44 | mse  0.44223
| epoch  10 |   500/  643 batches | lr 0.00 | ms/batch  5.61 | loss  0.36 | mse  0.35916
| epoch  10 |   600/  643 batches | lr 0.00 | ms/batch  4.76 | loss  0.46 | mse  0.45657
-----------------------------------------------------------------------------------------
| end of epoch  10 | time:  5.04s | valid loss  1.92 | valid mse  1.92142
-----------------------------------------------------------------------------------------
=========================================================================================
| End of training | test loss  1.82 | test mse  1.81831
=========================================================================================
```

```
!chmod +x sepens/make_models_perpat.sh
!./sepens/make_models_perpat.sh
```

```
| Generated 0/4507 timesteps
| Generated 100/4507 timesteps
| Generated 200/4507 timesteps
| Generated 300/4507 timesteps
| Generated 400/4507 timesteps
| Generated 500/4507 timesteps
| Generated 600/4507 timesteps
| Generated 700/4507 timesteps
| Generated 800/4507 timesteps
| Generated 900/4507 timesteps
| Generated 1000/4507 timesteps
| Generated 1100/4507 timesteps
| Generated 1200/4507 timesteps
| Generated 1300/4507 timesteps
| Generated 1400/4507 timesteps
| Generated 1500/4507 timesteps
| Generated 1600/4507 timesteps
| Generated 1700/4507 timesteps
| Generated 1800/4507 timesteps
| Generated 1900/4507 timesteps
| Generated 2000/4507 timesteps
| Generated 2100/4507 timesteps
| Generated 2200/4507 timesteps
| Generated 2300/4507 timesteps
| Generated 2400/4507 timesteps
| Generated 2500/4507 timesteps
| Generated 2600/4507 timesteps
| Generated 2700/4507 timesteps
| Generated 2800/4507 timesteps
| Generated 2900/4507 timesteps
| Generated 3000/4507 timesteps
| Generated 3100/4507 timesteps
| Generated 3200/4507 timesteps
| Generated 3300/4507 timesteps
| Generated 3400/4507 timesteps
| Generated 3500/4507 timesteps
| Generated 3600/4507 timesteps
| Generated 3700/4507 timesteps
| Generated 3800/4507 timesteps
| Generated 3900/4507 timesteps
| Generated 4000/4507 timesteps
| Generated 4100/4507 timesteps
| Generated 4200/4507 timesteps
| Generated 4300/4507 timesteps
| Generated 4400/4507 timesteps
```

Evaluation

These evaluation "inference_" sh scripts from the paper's git repo will help generate all of the model-inferenced datasets so that we can compare all of these results to the ground truth labels at once instead of reloading the ground truth over and over again. The calc_auroc.py is where we get the accuracy metric AUROC (Area Under the Receiver Operating Characteristic Curve) results at the bottom, which measures a model's ability to differentiate between positive and negative classes by plotting the true positive rate against the false positive rate at various classification thresholds.

```
!chmod +x sepens/inference_poolmodels.sh
!./sepens/inference_poolmodels.sh
```

```
| Generated 1900/4507 timesteps
| Generated 2000/4507 timesteps
| Generated 2100/4507 timesteps
| Generated 2200/4507 timesteps
| Generated 2300/4507 timesteps
| Generated 2400/4507 timesteps
| Generated 2500/4507 timesteps
| Generated 2600/4507 timesteps
| Generated 2700/4507 timesteps
| Generated 2800/4507 timesteps
| Generated 2900/4507 timesteps
| Generated 3000/4507 timesteps
| Generated 3100/4507 timesteps
| Generated 3200/4507 timesteps
| Generated 3300/4507 timesteps
| Generated 3400/4507 timesteps
| Generated 3500/4507 timesteps
| Generated 3600/4507 timesteps
| Generated 3700/4507 timesteps
| Generated 3800/4507 timesteps
| Generated 3900/4507 timesteps
| Generated 4000/4507 timesteps
| Generated 4100/4507 timesteps
| Generated 4200/4507 timesteps
| Generated 4300/4507 timesteps
| Generated 4400/4507 timesteps
| Generated 4500/4507 timesteps
EncNum 11460, Model 11460
| Generated 0/4507 timesteps
| Generated 100/4507 timesteps
| Generated 200/4507 timesteps
| Generated 300/4507 timesteps
| Generated 400/4507 timesteps
| Generated 500/4507 timesteps
| Generated 600/4507 timesteps
| Generated 700/4507 timesteps
| Generated 800/4507 timesteps
| Generated 900/4507 timesteps
| Generated 1000/4507 timesteps
| Generated 1100/4507 timesteps
| Generated 1200/4507 timesteps
| Generated 1300/4507 timesteps
| Generated 1400/4507 timesteps
```

```
!python3 sepens/MEDIUM_grow_ensemble_perrone.py 0 | tee logs/grow_ensemble.log
!tail -n1 logs/grow_ensemble.log > new_ensemble.py
!sed "s/ /\n/g" new_ensemble.py | sed 's/[^0-9]*//g' | sed -r '/^\s*$/d' > new_ensemble.lst
```

```
INFO: loading timelines and predictions.
ns=1, ss=4
.
no.preds=5
no.labels=24887
INFO: model predictions compiled.
ns_s=1, ss_s=4
1 1.9651270252888657 septic
2 1.9318387374690007 septic
ns = 0 , ss = 2
[(11924, 1.9651270252888657), (12302, 2.3985392957808527)]
-----------
11924
12302
-----------
ensemble = {11924, 12302, }
[(2, 1.9318387374690007), (1, 1.9651270252888657)]
```

```
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/new_ensemble.py" "/content/sepens/new_ensemble.py"
!cp "/content/drive/My Drive/DL4H_Final_Files/sepens_folder/ensemble_predictions.py" "/content/sepens/ensemble_predic
```

```
!chmod +x sepens/inference_fullmodel.sh
!./sepens/inference_fullmodel.sh
```

```
EncNum 12625
| Generated 0/5424 timesteps
| Generated 100/5424 timesteps
```

```
| Generated 200/5424 timesteps
| Generated 300/5424 timesteps
| Generated 400/5424 timesteps
| Generated 500/5424 timesteps
| Generated 600/5424 timesteps
| Generated 700/5424 timesteps
| Generated 800/5424 timesteps
| Generated 900/5424 timesteps
| Generated 1000/5424 timesteps
| Generated 1100/5424 timesteps
| Generated 1200/5424 timesteps
| Generated 1300/5424 timesteps
| Generated 1400/5424 timesteps
| Generated 1500/5424 timesteps
| Generated 1600/5424 timesteps
| Generated 1700/5424 timesteps
| Generated 1800/5424 timesteps
| Generated 1900/5424 timesteps
| Generated 2000/5424 timesteps
| Generated 2100/5424 timesteps
| Generated 2200/5424 timesteps
| Generated 2300/5424 timesteps
| Generated 2400/5424 timesteps
| Generated 2500/5424 timesteps
| Generated 2600/5424 timesteps
| Generated 2700/5424 timesteps
| Generated 2800/5424 timesteps
| Generated 2900/5424 timesteps
| Generated 3000/5424 timesteps
| Generated 3100/5424 timesteps
| Generated 3200/5424 timesteps
| Generated 3300/5424 timesteps
| Generated 3400/5424 timesteps
| Generated 3500/5424 timesteps
| Generated 3600/5424 timesteps
| Generated 3700/5424 timesteps
| Generated 3800/5424 timesteps
| Generated 3900/5424 timesteps
| Generated 4000/5424 timesteps
| Generated 4100/5424 timesteps
| Generated 4200/5424 timesteps
| Generated 4300/5424 timesteps
| Generated 4400/5424 timesteps
| Generated 4500/5424 timesteps
| Generated 4600/5424 timesteps
| Generated 4700/5424 timesteps
| Generated 4800/5424 timesteps
| Generated 4900/5424 timesteps
| Generated 5000/5424 timesteps
| Generated 5100/5424 timesteps
| Generated 5200/5424 timesteps
| Generated 5300/5424 timesteps
| Generated 5400/5424 timesteps
EncNum 12302
| Generated 0/5363 timesteps
```

```
!chmod +x sepens/inference_ensmodels.sh
!./sepens/inference_ensmodels.sh
```

```
EncNum 12625, Model 11924
EncNum 12625, Model 12302
EncNum 12302, Model 11924
EncNum 12302, Model 12302
EncNum 11924, Model 11924
EncNum 11924, Model 12302
EncNum 13700, Model 11924
EncNum 13700, Model 12302
EncNum 11460, Model 11924
EncNum 11460, Model 12302
```

```
!python3 sepens/calc_auroc.py
```

```
Interval: -4.250000 to -3.750000
full model: 0.5
uniform:    0.0
weighted:   0.0
```

```
Interval: -8.250000 to -7.750000
full model: 0.5
uniform:    0.0
weighted:   0.0
Interval: -12.250000 to -11.750000
full model: 0.5
uniform:    0.0
weighted:   0.0
Interval: -12.250000 to -7.750000
full model: 0.5
uniform:    0.0
weighted:   0.0
Interval: -24.250000 to -11.750000
full model: 0.5
uniform:    0.0
weighted:   0.0
```

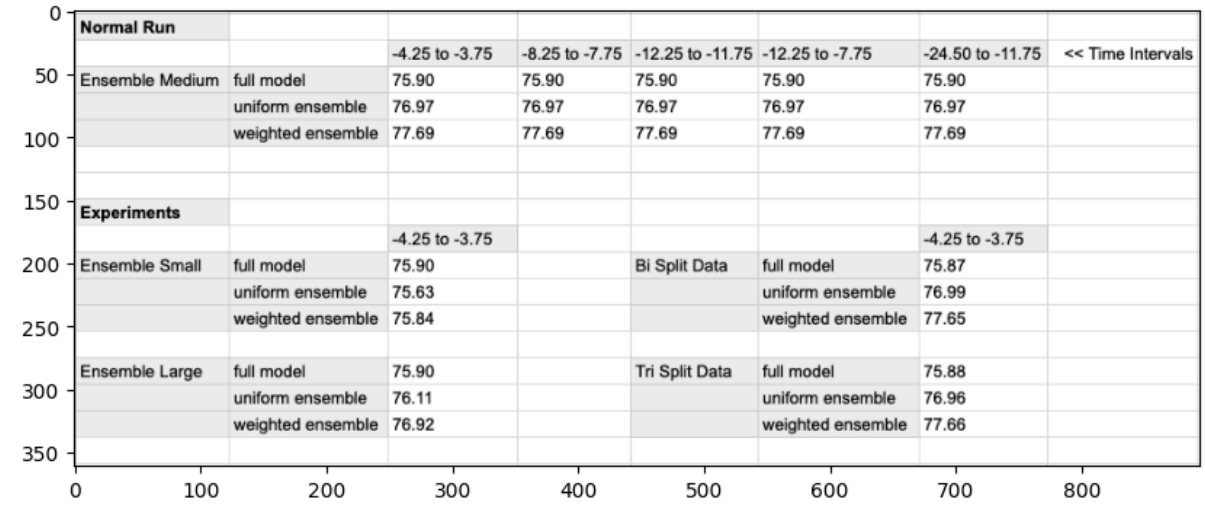
Results

Table Of Results:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.imshow(plt.imread('results.png'))
```

```
<matplotlib.image.AxesImage at 0x7cb5c4bf5ba0>
```



Normal Run							
		-4.25 to -3.75	-8.25 to -7.75	-12.25 to -11.75	-12.25 to -7.75	-24.50 to -11.75	<< Time Intervals
Ensemble Medium	full model	75.90	75.90	75.90	75.90	75.90	
	uniform ensemble	76.97	76.97	76.97	76.97	76.97	
	weighted ensemble	77.69	77.69	77.69	77.69	77.69	
Experiments							
		-4.25 to -3.75				-4.25 to -3.75	
Ensemble Small	full model	75.90		Bi Split Data	full model	75.87	
	uniform ensemble	75.63			uniform ensemble	76.99	
	weighted ensemble	75.84			weighted ensemble	77.65	
Ensemble Large	full model	75.90		Tri Split Data	full model	75.88	
	uniform ensemble	76.11			uniform ensemble	76.96	
	weighted ensemble	76.92			weighted ensemble	77.66	

Reflection on Paper's Hypothesis and My Results:

The paper's original hypothesis had been that the ensemble of patient-specific models could outperform the base full model. In the original paper, the full model averaged around 88.2 and the ensemble averaged uniform 90.5 and weighted 91.3 for the AUROC scores, indicating that the ensemble did indeed outperform the full model by a little bit. Though my own results were evaluated on a sub-section of data (1/8th of data) and thus not run for as long or on as much data, which led to overall lower accuracy results, my results did portray a similar trend. The ensembles outperformed the full model, with the weighted ensemble outpeforming the uniform ensemble, all by just a little bit. This was still pleasant to see that the principle seemed to be slightly visible even under a slice of the data.

Experiments and Ablations:

The original paper did not discuss how it arrived at its chosen ensemble size. I dug through its code to find where the ensemble growing algorithm was, which gave me a lever that controlled how large the ensemble was able to get when testing out different patient-specific models. I decided to a test an ablation of different ensemble sizes to see how the components of the model contributed to the experiment.

Their original ensemble averaged a size of 41 models, so I tested the MEDIUM_grow.py file with that size as the normal/original experiment. As my other experiments with this, I tested a size limit of 20 with the SMALL_grow.py file and a limit of 150 with the LARGE_grow.py file. I found from my tests that this didn't affect the ensembles a whole lot in their accuracy, and if anything made them a bit more inaccurate, which gives credence to the chosen 41 number for the amount of models that was used in the original paper's ensemble. Because this experiement was focused on the ensemble size, it of course did not change the outcome of the full base model's results.

The original paper had additionally split up and sampled the data to assign to the models for training. They were grouping 4 patients at a time to provide as data, but did not explain too much on that number, nor did they try to assess the experiment with different numbers. I intended to assess the performance on different sampling sizes, like 2 patients and 3 patients, and see how the model's acccuracy changes. Visualized in the data section above in this notebook, I created data files with these patient splits so I could run the pre-processing and run different executions of the training with these newly generated training/testing pairs.

Trying this out showed me that there wasn't much of a change in accuracy for any of the models that were significant. The full base model perhaps became a little less accurate in the bi-split because it was too many patients to generalize too, while the ensemble maybe thrived under these conditions to sample more patients and find better fits for making predictions on the remainder of the data. The tri-split seemed to be just negligible variation of a similar outcome to the original experiment.

Discussion

Implications of Experiment Results:

My experiment results showed me that the original paper was correct in the ensemble size they went with, as it maximized its prediction accuracy of the ensemble models. I also found that the split that the original paper applied was appropriate to use as their sampling technique, and that my proposed splits to test out of 2-way and 3-way did not affect the outcome too heavily. I do think that this lack of impact may have also been due to my usage of only a portion of the data (1/8th), so possibly with the whole data set we would have seen more drift between the full model and te ensemble, as patient-specific models maybe get an advantage with the patients available to choose from to build the ensemble.

Reproducibility of the Paper:

The paper is mostly reproducible, especially its code segments, as I was able to execute all parts of it successfully from their repo. I think achieving the same accuracy and being able to test all portions of the experiment is just a matter of GPU power andd available time to run the GPU for, but other than that, this is quite reproducible. A CPU option would certainly make this more accessible to people, as that allows them to run these tests for as long as neeeded, but without any concern of cloud engines locking out/shutting down/timing out because of running for too long. The data size is what made this paper's reproducibility strained, as it required an extremely long time to complete all the training using Colab's GPU. But with more time given, this could be fully reproducible.

What Was Easy: Using the code from the paper's git repo was relatively easy was I figured out how each individual file worked and was related to each other. After learning how to execute those files in Google Colab, I was able to build a smooth running system for myself to test different experiments. Loading the data and pre-processing it was especially a breeze thanks to the git repository.

What Was Difficult:

The GPU was a severe struggle for me, as it is not strong enough to complete things in a manner that allows me to execute the whole experiment and my other experiments in a timely manner. There was too much data to train on, so I had to cut down on that, and battle with Google Colab on getting kicked out of GPU access while I was working on things due to the amount of time needed to execute. I had to learn how to adapt by keeping my machine open and logged in, avoiding deactivation and doing proper testing to figure out what subset of data would be doable for my own testing as well as demo execution in this notebook.

Recommendations to the Original Authors for Improving Reproducibility:

I think it would help with reproducibility to provide an option to run all of these experiments using a CPU, that way it is more accessible to users looking to test (even if the tests take an extremely long time to complete). Other than that, they did a great job providing helpful files in their repo to use for executing all of the training and evaluation. There is not too much we can do about the amount of data there is to train with and the GPU size that is available to others, nor the amount of titme that would be needed to train on data, so they did the best they could to make this mostly adaptable to regular people/academics.