Landmark Mortality with NN

Bryan Cai

MIT

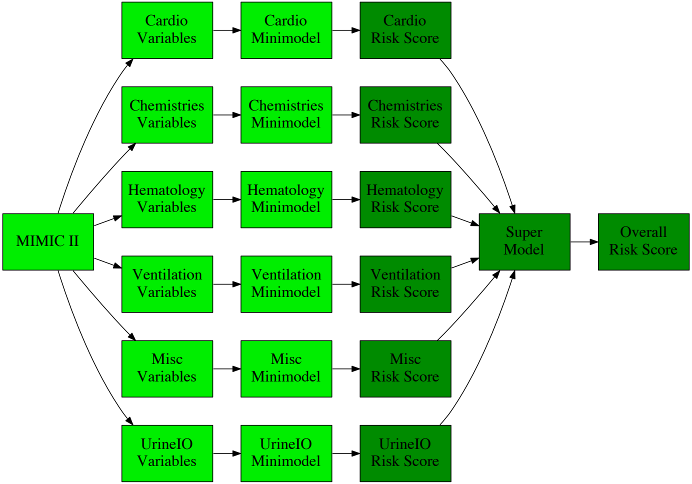
November 16, 2016

Table of Contents

Overview

- Goal was to predict mortality based on data taken from the ICU
  - Mortality was measured 30 days from the ICU entry date
- Grouped related variables together
  - Cardio, Chemistries, Hematology, Ventilation, UrineIO, and Miscellaneous groupings
- Create mini models and calculate a separate mortality score for each patient using the groups of variables
- Consolidate the models into a super model

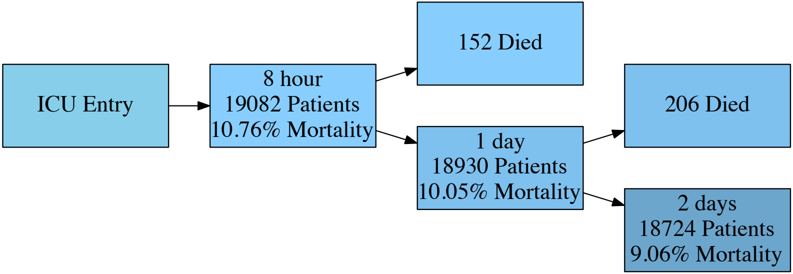
Overview


MIMIC II
Cardio Variables
Cardio Minimodel
Cardio Risk Score
Chemistries Variables
Chemistries Minimodel
Chemistries Risk Score
Hematology Variables
Hematology Minimodel
Hematology Risk Score
Ventilation Variables
Ventilation Minimodel
Ventilation Risk Score
Misc Variables
Misc Minimodel
Misc Risk Score
UrineIO Variables
UrineIO Minimodel
UrineIO Risk Score
Super Model
Overall Risk Score

Landmarking

- This procedure was performed for each distinct landmark time
  - 8 hours, 1 day, and 2 day from ICU entry time
- The model for landmark time T aims to predict mortality for patients who have survived until time T based on data up to time T
- Data retrieval and manipulation was done using Python
- Analysis was done using R

Landmarking



Data Filtering

- For each landmark time T, we used the data where:
  - The patient was still alive at time T, and
  - The data was recorded before time T
- Deleted variables where less than 5% of patients had measurements taken

Data Processing

- For each variable v and each patient, a set of summary variables were created:
  - Calculated maximum, minimum, mean, median, median absolute deviation, number of measurements, and an indicator variable for whether any measurements took place
- Imputed missing data using mean

Table of Contents

Neural Network Structure

- Tried several different network architectures
  - Input $\rightarrow$ 500 $\rightarrow$ 100 $\rightarrow$ output
  - Input $\rightarrow$ 1000 $\rightarrow$ 500 $\rightarrow$ output
  - Input $\rightarrow$ 500 $\rightarrow$ 700 $\rightarrow$ 100 $\rightarrow$ output
  - Input $\rightarrow$ 100 $\rightarrow$ output
- Used ReLU as our activation function for the hidden layers
- Used the sigmoid function for the output layer

Training and Loss Function

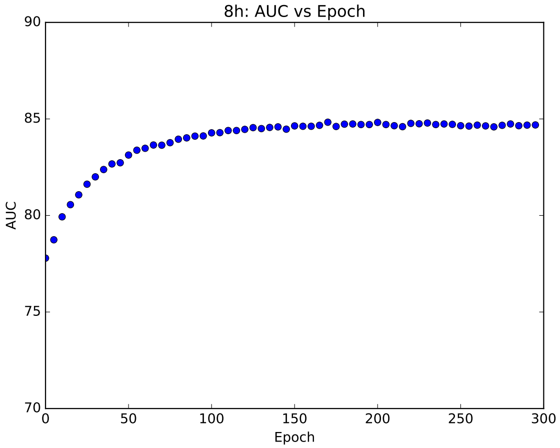
- Applied dropout before the output layer to address overfitting
- Used binary cross entropy as our loss function
- The model is implemented in Torch
- Used Lutorpy to embed Torch code into Python for easy data manipulation

Table of Contents

Baseline

40,000	LASSO	SVM
AUC	83.4%	80.7%

Overall



Overall AUC: 84.5%

Two Layered

40,000	NN	LASSO	SVM
NN	82.2%	**83.1%**	79.3%
LASSO	77.4%	76.7%	72.7%
SVM	78.2%	79.2%	75.2%

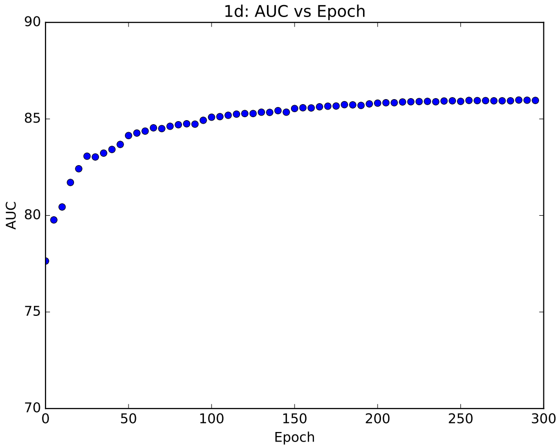
Highest Achieved: 85.0% (NN-NN)

Table of Contents

Baseline

40,000	LASSO	SVM
AUC	85.0%	81.3%

Overall



Overall AUC: 85.9%

Two Layered

40,000	NN	LASSO	SVM
NN	83.9%	86.3%	79.7%
LASSO	84.8%	83.8%	78.7%
SVM	83.9%	**86.4%**	77.2%

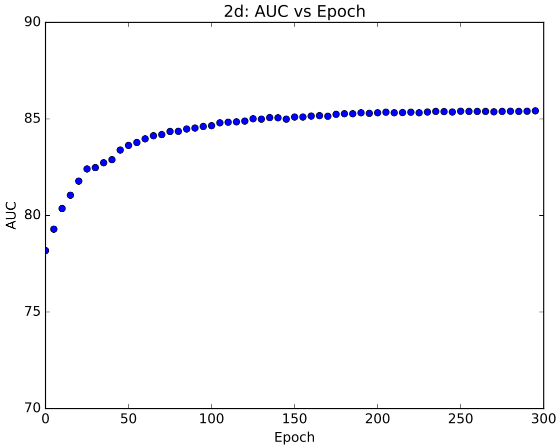
Highest Achieved: 88.7% (LASSO-NN)

Table of Contents

Baseline

40,000	LASSO	SVM
AUC	85.1%	83.3%

Overall



Overall AUC: 85.4%

Two Layered

40,000	NN	LASSO	SVM
NN	**85.6%**	82.5%	81.6%
LASSO	83.2%	83.9%	79.2%
SVM	81.2%	82.1%	76.5%

Highest Achieved: 87.7% (NN-NN)