Lab8: Mortality Prediction Challenge

Intro to Data Mathematics 2021

Jared Gridley

Overview

Every day, doctors and nurses collect a lot of information about patients by asking questions and using adapted tools (stethoscope, syringe, sensors, etc.). This data is very useful to monitor health state, diagnose and choose treatments. It can also be used for statistical predictive analysis of how the patient will progress. Analysts at the *GetUHealthy Hospital* are currently using the Mean Method to predict mortality. The first part of this lab introduces you to the Mean Method using data from the Medical Information Mart for Intensive Care (MIMIC). The second part involves you in an online challenge to improve a system for predicting mortality of patients in the hospital's Intensive Care Unit (ICU).

Mortality Prediction Challenge

The main question of this challenge is: *How to predict the survival of a patient in the ICU given his or her medical record?* More specifically, you are asked to predict whether or not a patient will die during their stay at the hospital. Healthcare organizations use risk models one to help determine high risk patients that may need specialized care. Models like this are being used at Mount Sinai hospitals in New York City to predict if COVID-19 patients are likely to need ICU care and ventilation. They are also used to help determine which COVID-19 patients may safely recuperate at home. Medical experts use techniques like those in this lab, except they are based on more sophisticated deep learning models called *neural networks*.

Data

Data Description

The MIMIC training dataset contains information on 79,999 patients, represented by *categorical*, *binary* and *numerical* features (also called "variables"). Those features are for instance age, gender, ethnicity, marital status, as well as medical data such as blood pressure rate or glucose rate. There are a total of 342 variables. For the code solutions for Mean Method and LDA here, we just use the 49 numeric features. But feel free to use more of the variables in your creative analysis.

The class (or label) to be predicted is a binary variable telling if the patient died or not during while in the hospital. Fortunately for the patients, most of them don't die. Unfortunately for *us*, that leads to a class imbalance.

Data Preparation

This section reads in the training data and creates the "internal" training and validation set. A validation set acts like a test set. It has known class labels, and we use it to determine how good our classifier are. We call it a validation set to distinguish it from the "external" test set. The big difference is that you don't know the labels of this external test set. *Your job is to predict the labels of the external test set to enter the challenge.*

Your first step is to prepare the data. For the basic entry provided here, we only use numeric features that contain age and various biochemical indicators. *We do not scale.* For your entry, you can decide to use more features or scale. Just remember that you may need to convert non-numeric features to numbers first. Read the comments and the contest documentation to understand the files that are provided.

```
# Read in mortality data
mortdata <- read.csv("~/MATP-4400/data/mimic_synthetic_train.data", sep=' ', header=F)[ ,-1] # exclude

# read in the labels of the training data
labels <- factor(read.csv("~/MATP-4400/data/mimic_synthetic_train.solution", header=F)[ ,1], levels=c(0

#read in the feature types
feat.types <- read.csv("~/MATP-4400/data/mimic_synthetic_feat.type", stringsAsFactors = F, header=F)[,1]
# read in the feature names
cnames <- read.csv("~/MATP-4400/data/mimic_synthetic_feat.name", stringsAsFactors = F, header=F)[,1]

#Figure out which features are numerical
bool.feats <- feat.types=='Numerical'

mortdata <- mortdata[ ,bool.feats][ ,3:51] # only keep age and biochemical indicators
colnames(mortdata) <- cnames[bool.feats][3:51] # name the columns
```

The data was randomly divided into two sets consisting of 90% training and 10% validation. We'll use the validation set to estimate how well the classifier may work on future data.

```
# Split the data into training and testing sets
# train.size will be the number of data in the training set
n <- nrow(mortdata)
train.size <- ceiling(n*0.9)

#Set random seed to ensure reproducibility
set.seed(300)
trainID <- sample(n,train.size)

# For testing purposes...
head(trainID)
```

```
## [1] 66325 32700 59049 32897  4006  1312
```

```
#The training data is just the training rows
morttrain <- mortdata[trainID, ]

# Using -train gives us all rows except the training rows.
mortval <- mortdata[-trainID, ]
trainclass <- labels[trainID]
valclass <- labels[-trainID]

#We leave off the last column (the class label) to get the matrices of features
trainmatrix <- as.matrix(morttrain)
valtmatrix <- as.matrix(mortval)

# Let's look at the class labels of the training data
summary(trainclass)
```

```
## lived  died
## 69489  2511
```

```r
summary(valclass)
```

```
## lived  died
##  7713   286
```

Exercise 1

Predict ICU Mortality using the Mean Method.

Construct a classifier on the training set using the Mean Method. This has been done for you.

```r
Q <- trainmatrix
labels <- trainclass

# Make a vector y of class with 1 if M or -1 if B
y <- matrix(1,nrow=length(labels))
y[labels=='lived',1] <- -1

# Make Cplus the matrix containing the positive data
Cplus <- Q[y==1,]

# Make Cplus the matrix containing the positive data
Cneg=Q[y==-1,]

Mplus<-colMeans(Cplus)
Mneg<-colMeans(Cneg)
wMM<-(Mplus-Mneg)
wMM<-wMM/sqrt(sum(wMM^2)) # w/norm(w)

# Calculate threshold t
tMM <- ((Mplus+Mneg)%*%wMM)/2
tMM<-as.numeric(tMM) # make sure t is a scalar, not a matrix

tMM
```

```
## [1] 272.0359
```

Predict the training set using the Mean Method: Calculate the *Class 1* accuracy, the *Class -1* accuracy, and the *balanced* accuracy on the training set.

```r
wMM.X <- trainmatrix %*% wMM
wMM.y <- as.matrix(wMM.X[, 1] < tMM )

wMM.X.pos <- as.matrix(wMM.X[wMM.y == TRUE,] )
wMM.X.neg <- as.matrix(wMM.X[wMM.y == FALSE,] )

train.pred <- as.matrix(c(nrow(wMM.X.pos), nrow(wMM.X.neg)))
rownames(train.pred) <- c("lived", "died")
train.class.data <- as.matrix(summary(trainclass))

confusion.matrix <- t(cbind(train.class.data, train.pred))
# Calculate the accuracy of each class using the entries of the confusion matrix
accneg <-confusion.matrix[1,1]/(confusion.matrix[1,1]+confusion.matrix[1,2])
accplus <-confusion.matrix[2,2]/(confusion.matrix[2,1]+confusion.matrix[2,2])
# Calculate the balanced accuracy
```

```
accbal<-(accplus+accneg)/2

accneg
```

```
##    lived
## 0.965125
```

```
accplus
```

```
##      died
## 0.3649861
```

```
accbal
```

```
##      died
## 0.6650556
```

On the train dataset, the mean method classified 96.5% of the positive training data correctly, but only classified 36.5% of the negative training data. The Balanced training accuracy was 66.5%. So overall this is not what we want, however it was somewhat expected because we know that the mean method doesn't work as well on data that isn't evenly split.

Predict the validationset using the Mean Method: Calculate the *Class 1* accuracy, the *Class -1* accuracy, and the *balanced* accuracy on the validation set.

```
wMM.X <- valtmatrix %*% wMM
wMM.y <- as.matrix(wMM.X[, 1] < tMM )

wMM.X.pos <- as.matrix(wMM.X[wMM.y == TRUE,] )
wMM.X.neg <- as.matrix(wMM.X[wMM.y == FALSE,] )

val.pred <- as.matrix(c(nrow(wMM.X.pos), nrow(wMM.X.neg)))
rownames(train.pred) <- c("lived", "died")
val.class.data <- as.matrix(summary(valclass))

confusion.matrix <- t(cbind(val.class.data, val.pred))
# Calculate the accuracy of each class using the entries of the confusion matrix
accneg <-confusion.matrix[1,1]/(confusion.matrix[1,1]+confusion.matrix[1,2])
accplus <-confusion.matrix[2,2]/(confusion.matrix[2,1]+confusion.matrix[2,2])
# Calculate the balanced accuracy
accbal<-(accplus+accneg)/2

accneg
```

```
##     lived
## 0.9642455
```

```
accplus
```

```
##      died
## 0.3731716
```

```
accbal
```

```
##      died
## 0.6687086
```

On the test dataset, the mean method classified 96.4% of the positive testing data correctly, but only classified 37.3% of the negative testing data. The Balanced training accuracy was 66.9%. So overall this closely

resembled the accuracy results for the training data, which is good because it means that our test set is a good representation of the dataset. However the mean method is not so good on this dataset because it is not evenly split

Exercise 2

Construct a classifier on the training set using the **LDA Method**.

```
# Make a train.df and test.df dataframes combining labels and features
train.df<-data.frame(trainmatrix, class=as.factor(trainclass))
test.df<-data.frame(valtmatrix, class=as.factor(valclass))

lda.fit <- lda(class~., train.df, prior = c(1,1)/2)
w<-lda.fit$scaling
thresh <- ((lda.fit$means[1,] +lda.fit$means[2,])/2)%*%lda.fit$scaling
```

Predict the training set: Calculate the *Class 1* (Died) accuracy, the *Class -1* (Lived) accuracy, and the *balanced* accuracy on the training set.

```
train.pred <- predict(lda.fit,train.df)$class

confusion.matrix <- table(trainclass, train.pred)
confusion.matrix
```

```
##           train.pred
## trainclass lived  died
##      lived 51584 17905
##      died    738  1773
```

```
accneg <-confusion.matrix[1,1]/(confusion.matrix[1,1]+confusion.matrix[1,2])
accplus <-confusion.matrix[2,2]/(confusion.matrix[2,1]+confusion.matrix[2,2])
# Calculate the balanced accuracy
accbal<-(accplus+accneg)/2

accneg
```

```
## [1] 0.7423333
```

```
accplus
```

```
## [1] 0.7060932
```

```
accbal
```

```
## [1] 0.7242133
```

On the train dataset, the LDA method classified 74.2% of the positive training data correctly, but only classified 70.6% of the negative training data. The Balanced training accuracy was 72.4%. So overall this is better than the mean method for the training data so far because even though it dod not get as high of accuracy with the positive training data, the overall is better.

Predict the validation set using the LDA method: Calculate the *Class 1* (Died) accuracy, the *Class -1* (Lived) accuracy, and the *balanced* accuracy on the validation set.

```
test.pred <- predict(lda.fit,test.df)$class

confusion.matrix <- table(valclass, test.pred)
confusion.matrix
```

```
##         test.pred
```

```
## valclass lived died
##    lived  5754 1959
##    died     73  213
```

```
accneg <-confusion.matrix[1,1]/(confusion.matrix[1,1]+confusion.matrix[1,2])
accplus <-confusion.matrix[2,2]/(confusion.matrix[2,1]+confusion.matrix[2,2])
# Calculate the balanced accuracy
accbal<-(accplus+accneg)/2

accneg
```

```
## [1] 0.7460132
```

```
accplus
```

```
## [1] 0.7447552
```

```
accbal
```

```
## [1] 0.7453842
```

Discuss how the LDA and Mean Method Classifiers are performing. Discuss the scores on the training and validation sets and compare. Which method is working better?
Overall, the LDA method classifier performed better than the mean method. This was generally expected because we knew that our data would not be evenly split between those taht lived and those that died, so we could expect that the mean method would not work as well. On the train dataset, the mean method classified 96.5% of the positive training data correctly, but only classified 36.5% of the negative training data. On the test dataset, the mean method classified 96.4% of the positive testing data correctly, but only classified 37.3% of the negative testing data. The Balanced training accuracy was 66.9%. On the train dataset, the LDA method classified 74.2% of the positive training data correctly, but only classified 70.6% of the negative training data. The Balanced training accuracy was 72.4%. On the test dataset, the LDA method correctly classified 74.6% of the positive data, 74.5% of the negative data, which gave it an overall accuracy of 74.5%. which is significanly better than the mean method.

Exercise 3

Make a model to predict mortality using R any way you like. *Use your creativity to make a better predictive model!* Can you use more data? Can you use different features? Can you use a different modeling algorithm? Would scaling help (don't forget to scale test and validation the same)?

You can get three points for making a different model. Earn up to 3 points extra credit for being creative and trying different things.

```
# For my creative analysis I want to try out using bayes theorum (I read an article about it, so wanted
model <- naiveBayes(class~., data = train.df)
train.bayes.pred <- predict(model, train.df)
```

Predict the training set using your method. Calculate the *Class 1* (Died) accuracy, the *Class -1* (Lived) accuracy, and the *balanced* accuracy on the training set.

```
# For my creative analysis I want to try out using bayes theorum (I read an article about it, so wanted
confusion.matrix <- table(trainclass, train.bayes.pred)
confusion.matrix
```

```
##           train.bayes.pred
## trainclass lived  died
##      lived 63102  6387
##      died   1722   789
```

```
accneg <-confusion.matrix[1,1]/(confusion.matrix[1,1]+confusion.matrix[1,2])
accplus <-confusion.matrix[2,2]/(confusion.matrix[2,1]+confusion.matrix[2,2])
# Calculate the balanced accuracy
accbal<-(accplus+accneg)/2

accneg
```

```
## [1] 0.9080862
```

```
accplus
```

```
## [1] 0.3142174
```

```
accbal
```

```
## [1] 0.6111518
```

Predict the validation set using the your method. Calculate the *Class 1* accuracy, the *Class -1* accuracy, and the *balanced* accuracy on the validation set.

```
test.bayes.pred <- predict(model, test.df)

confusion.matrix <- table(valclass, test.bayes.pred)
confusion.matrix
```

```
##         test.bayes.pred
## valclass lived died
##    lived  7030  683
##    died    184  102
```

```
accneg <-confusion.matrix[1,1]/(confusion.matrix[1,1]+confusion.matrix[1,2])
accplus <-confusion.matrix[2,2]/(confusion.matrix[2,1]+confusion.matrix[2,2])
# Calculate the balanced accuracy
accbal<-(accplus+accneg)/2

accneg
```

```
## [1] 0.9114482
```

```
accplus
```

```
## [1] 0.3566434
```

```
accbal
```

```
## [1] 0.6340458
```

This was not a significant improvement from the mean method and LDA. I was hoping that it would perfom better than an overall accuracy of around 61 in the training set and 61 in the test set because I read that bayes theorum was often used for disease prediction. However, this was just a simple bayes implementation, I will still try to improve it with Laplace smoothing.

```
model.laplace <- naiveBayes(class~., data = train.df, laplace = 3)
train.bayeslp.pred <- predict(model, train.df)
test.bayeslp.pred <- predict(model, test.df)
```

Testing on the training data

```
confusion.matrix <- table(trainclass, train.bayeslp.pred)
confusion.matrix
```

```
##           train.bayeslp.pred
```

```
## trainclass lived  died
##      lived 63102  6387
##      died   1722   789
```

```
accneg <-confusion.matrix[1,1]/(confusion.matrix[1,1]+confusion.matrix[1,2])
accplus <-confusion.matrix[2,2]/(confusion.matrix[2,1]+confusion.matrix[2,2])
# Calculate the balanced accuracy
accbal<-(accplus+accneg)/2

accneg
```

```
## [1] 0.9080862
```

```
accplus
```

```
## [1] 0.3142174
```

```
accbal
```

```
## [1] 0.6111518
```

Testing on the testing data

```
confusion.matrix <- table(valclass, test.bayeslp.pred)
confusion.matrix
```

```
##         test.bayeslp.pred
## valclass lived died
##    lived  7030  683
##    died    184  102
```

```
accneg <-confusion.matrix[1,1]/(confusion.matrix[1,1]+confusion.matrix[1,2])
accplus <-confusion.matrix[2,2]/(confusion.matrix[2,1]+confusion.matrix[2,2])
# Calculate the balanced accuracy
accbal<-(accplus+accneg)/2

accneg
```

```
## [1] 0.9114482
```

```
accplus
```

```
## [1] 0.3566434
```

```
accbal
```

```
## [1] 0.6340458
```

Still no significant change in the results that would make it better than LDA. Next, I am found a Datacamp tutorial on logistic regression so I'm going to try that.

```
fit <- vglm(class~., family = multinomial, data = test.df)
#summary(fit)

probabilities <- predict(fit, test.df, type="response")
predictions <- apply(probabilities, 1, which.max)
levels(train.df$class) <- c("lived", "died")
predictions[which(predictions==1)] <- levels(test.df$class)[1]
predictions[which(predictions==2)] <- levels(test.df$class)[2]

confusion.matrix <- table(valclass, predictions)
```

```
confusion.matrix
```

```
##         predictions
## valclass died lived
##    lived    9  7704
##    died    11   275
```

Calculating the Accuracy values:

```
accneg <-confusion.matrix[1,2]/(confusion.matrix[1,2]+confusion.matrix[1,1])
accplus <-confusion.matrix[2,1]/(confusion.matrix[2,2]+confusion.matrix[2,1])
# Calculate the balanced accuracy
accbal<-(accplus+accneg)/2

accneg
```

```
## [1] 0.9988331
```

```
accplus
```

```
## [1] 0.03846154
```

```
accbal
```

```
## [1] 0.5186473
```

This still isn't good overall, which was somewhat expected because of the way that linear regression works, it would probably perform better if there was more negative data beacuse it was able to get a very high accuracy on the positive data because it was most positives in the, but a very low accuracy on the negative data.

Exercise 4

- Compare performance of the Mean Method, LDA, and and your best shot model on the training and validation sets. Report how well the models do on the mortality data in terms of Class 1 accuracy, Class -1 accuracy, and balanced accuracy.

So far, the LDA model has had the best performance so far for fitting the data. It finished with a balanced accuracy of 0.7453842 (Pos: 0.7447552, Neg: 0.7460132). This was significantly better than the mean method and my other model when it came to correctly identifying the positive cases. The result for the mean method were a balanced accuracy of 0.6687086 (Pos: 0.9642455, Neg: 0.3731716). For my Bayes Theorum classifier the balanced accuracy was 0.6340458 (Pos: 0.3566434, Neg: 0.9114482). Even with Laplace smoothing, there wasn't a significant improvement, it was slightly worse than the mean method, but still comparable. For my creative Analysis, I applied Bayes Theorum and Logistic Regression to the data to predict ICU need. I found that bayes theorum did perform somewhat better than LDA in classifying negative predictions but performed significantly worse on the positive cases, giving it an overall accuracy below LDA. The same can be said for the logistic regression model. It performed exceptionally well on the negative cases but exceptionally bad on the positive cases, Overall 0.5104577 (Pos: 0.03846154, Neg: 0.9988331).

- Describe the predictive model you suggest for predicting ICU mortality on future patients. This could be any of the models that you have made in the previous sections. Explain why you selected this one.

I would select the LDA model to be used to predict ICU mortality beacuse it had the best results overall but also because there wasn't a large difference between the positive and negative accuracies A model like the Logistic Regression could be used to verify negative cases since it had a ver large degree of accuracy for negative cases.

Exercise 5

For this project and the final project, you will being participating in an online challenge.

The Lab8 challenge is the "To Be or Not To Be Mortality Challenge" at http://codalab.idea.rpi.edu/competitions/4

The challenge asks you to predict survival of the patients in `~/MATP-4400/data/mimic_synthetic_test.data`

- The winner is the person who gets the highest score on this data.
- The label of this data are 0 for lived and 1 for died.
- Note this is a change from the -1, 1 format given above.
- The predictions must be in a file. The file **must** be called `mimic_synthetic_test.csv`.
- The submission is made by zipping this file, into a file which can be called by any name.

This exercise walks you through how to make an entry using the mean method. You should submit the mean method as your first entry. Everyones submissions will be the same, so you will know if you successfully submitted an entry. Then you can try entering again using your best shot method.

Prepare testing

First, we need to get the external testing data and prepare it just like we did the training data.

```
# Read in the test data
mortdatatest <- read.csv("~/MATP-4400/data/mimic_synthetic_test.data", sep=' ', header=F)[ ,-1]

# Apply the same transformation to the test that we used on the training data.
mortdatatest <- as.matrix(mortdatatest[,bool.feats][ ,3:51]) # only keep age and numerical chemical dat
colnames(mortdatatest) <- cnames[bool.feats][3:51] # only keep age and numerical chemical data to avoid
```

Predict the external testing data

Now we predict the external testing data using the Mean method function made on the training data. (Note you may get a warning message here about object lengths but it is okay). We use the wMM and tMM to do the prediction.

```
# This code is provided for you
# For the test set,
# Make a prediction using the mean method
ypredtestMM<-sign(mortdatatest%*%wMM-tMM)
# deal with 0 case
ypredtestMM[ypredtestMM==0]<- 0

# Switch from 1 (Died) and -1 (Lived) classes to the numbers 1 and 0 classes for the contest entry
ypredtestMM <- recode(as.factor(ypredtestMM), "1"=1, "-1" =0)

#verify all predictions are 0 or 1
summary(as.factor(ypredtestMM))
```

```
##     0     1
## 12583  7418
```

Prepare the testing data predictions for submission.

Now we create the entry file and zip it up for submission.
'BEWARE the output in ypredtest must be numbers with values equal to 0 or 1.

```
# Here is the mean prediction file for submission to the website
```

```
write.table(ypredtestMM,file = "mimic_synthetic_test.csv", row.names=F, col.names=F)

#This automatically generates a compressed (zip) file
system("zip -u MMentry.csv.zip mimic_synthetic_test.csv")
```

Download the file `MMentry.csv.zip` to your laptop (using the RStudio **Export** command under the **More** menu item in **Files**) so you can enter the challenge.

Enter the challenge using the predictions in MMentry.csv.zip

Enter the Challenge and see your score.

- Go to http://codalab.idea.rpi.edu/competitions/4 and create an account using the **Sign Up** tab. Use your RCS ID as your username. Log in.
- Go to the **Participate** tab at: http://codalab.idea.rpi.edu/competitions/4
- Select **Submit/View Results**
- Describe your submission; make sure to mention your method
- Hit **Submit** to upload your file. Upload your `.zip` file from your computer.
- Hit **Refresh Status** to make sure that your submission was successful. *It can take a few minutes for the update to happen.*
- Look at the **View Scoring Output Log** tab to see how you did. You should see 60% balanced accuracy for the Mean Method.
- Go to the **Results** page to see how you did with respect to everyone.
- *If you see a negative number, then something went wrong and go back to the Output log to see what the problem was.*
- *Only your latest submission will count as your submission!*

Congratulations, you get three points if you successfully submit the Mean Method.

Write your score with the mean method here 0.600294326030.

Exercise 6

Now you will create your own entry into the challenge. Surely you can do better than the Mean Method!

Note the Mean Method was made up for educational purposes. You should never use it real life! LDA and many other methods will probably always do better.

- Create a `.zip` file with the name of your choice containing `mimic_synthetic_test.csv` which contains your test data predictions.
- Use the same procedure for submitting as you did for the Mean Method. You should predict either `0` or `1` for each test data point.
- Export your entry file from Rstudio using the **Export** function under **More** in the **Files** pane.
- Create a `.zip` file and submit to the challenge website.

- If your score (left-hand column of results table) is a *negative* number, then you made some mistake.
  - Check out `View scoring output log` on the website to see what might have gone wrong.
  - Common mistakes are to not have the correct number of predictions — for example, if you predict the training set instead of the testing set — or to predict labels other than `0` and `1`.

NOTE: Your entry filename **must** be: `mimic_synthetic_test.csv` and it **must be zipped to submit.**

```
# Read in the test data
mortdatatest <- read.csv("~/MATP-4400/data/mimic_synthetic_test.data", sep=' ', header=F)[ ,-1]

# Apply the same transformation to the test that we used on the training data.
mortdatatest <- as.matrix(mortdatatest[,bool.feats][ ,3:51]) # only keep age and numerical chemical dat
colnames(mortdatatest) <- cnames[bool.feats][3:51] # only keep age and numerical chemical data to avoid
```

```
test = as.data.frame(mortdatatest)

test.pred <- predict(lda.fit,test)
format.test.pred <- recode(as.factor(test.pred$class), "died"=1, "lived" =0)

write.table(format.test.pred,file = "mimic_synthetic_test.csv", row.names=F, col.names=F)

#This automatically generates a compressed (zip) file
system("zip -u MMentry.csv.zip mimic_synthetic_test.csv")
```

- Give your RCS ID/entry name and the score that you received on the website here:
- **RCS ID = gridlj Balanced Accuracy = 0.732320955500**

The winning entry gets 5 points extra-credit.

Extra Credit (3 points)

Machine learning classification models are being used to combat the COVID-19 pandemic. Find a published paper that does mortality prediction for COVID-19. (Google Scholar is a good place to look). Summarize the problem the papers solves, the data set that is used, the machine learning method used. and the results. Make sure to include the reference for the paper.

Problem: The algorithm proposed in this paper was designed to predict mortality rates in covid-19 patients as early as possible in order to allow for ealy intervention in treatment and reduce the probability that the patients who are the most at risk would be priortized in order to efficiently reduce the mortality rate. Development of the dataset: Medical information of patients between Jan 10, 2020 and Feb 18 2020 was used to develop the model. (The dataset excluded pregnant, breast-feeding, and patients younger than the age of 18). The model training and testing used only teh data from the final sample of inputs to the model to access the crucial biomarkers of disease severity. Machine Learning methods used: This study used a supervised XGBoost classifier. XGBoost is a multi-decision tree algorithm that was used to assess the importance of each feature. The performances of the model showed no improvement in area under the curve (AUC) scores when the number of top features increased to four. Hence, the number of key features was set to the following three: lactic dehydrogenase (LDH), lymphocytes and high-sensitivity C-reactive protein (hs-CRP)

Final results: The XGBoost machine learning model was able to predict mortality rates of patients more than 10 days in advance with 90% accuracy, which allowed for detection and early intervention to reduce the risk of mortality.

link: https://www.nature.com/articles/s42256-020-0180-7.pdf