Predicting 28-day mortality from the response to an initial session of prone positioning in intubated patients using Machine Learning

David M. Hannon

Introduction

This work seeks to build a machine learning model that will predict the 28-day mortality of a patient who is turned to the prone position due to significant respiratory dysfunction. The prone position (i.e. the 'face down' position) is used in patients with Acute Respiratory Distress Syndrome (ARDS), and when used appropriately can improve mortality in this cohort[1]. It is performed in 12-16 hour episodes, and can be (and often is) performed repeatedly[2].

Prognostication at an individual level is challenging[3]. Placing a patient in the prone position in logistically difficult and can lead to dangerous complications. Being able to predict mortality from a patients initial response to being placed in the prone position can inform decisions on whether the treatment should be repeated or whether other therapies such as Extra Corporeal Membrane Oxygenation (ECMO) should be considered.

The data

Data was gathered from the Electronic Health Record System (EHRS) that is used in the Intensive Care Unit (ICU) of University Hospital Galway (UHG). This specifically sought the records for adult patients who were invasively ventilated (i.e. a breathing tube was inserted and a ventilator used to breathe for the patient) and subsequently placed in the prone position at least once. Physiological parameters, blood test results, and ventilator readings were recorded. The patients included were admitted between 14/07/2013 and 20/03/2022.

The records were interrogated to determine which of several broad categories of lung injury the patient fell into. The resultant classifications were added, and are outlined in Table 1.

Abbreviation	**Pathology**
ARDSp	ARDS secondary to a direct pulmonary insult
ARDSexp	ARDS secondary to a non-pulmonary insult
Covid-19	ARDS secondary to Covid-19
Unknown	No clear underlying cause

Table 1: Classification of ARDS aetiology

The data for each patient was examined and key variables relating to the ventilatory status of the patient, the inflammatory state of the patient, and key demographic variables were isolated. These values were recorded at three key time-points. Baseline biochemistry and haematology values were noted at the nearest time-point preceding being placed in the prone position. One arterial blood gas (ABG) and corresponding ventilatory values were recorded from immediately before being turned to the prone position. The same set of values was recorded at the end of the session of prone positioning, prior to returning to the supine position. Finally, recordings were taken from within four hours of returning to the supine position.

Additional values were also calculated and added to the dataset at each time-point. This includes:

- the PF ratio (a measure of efficacy of oxygenation)
- the Aa gradient (a surrogate for pulmonary shunting)
- Body Mass Index (BMI)

The resultant dataframe has 42 variables. These, and the amount of data missing, is outlined in the following figure.

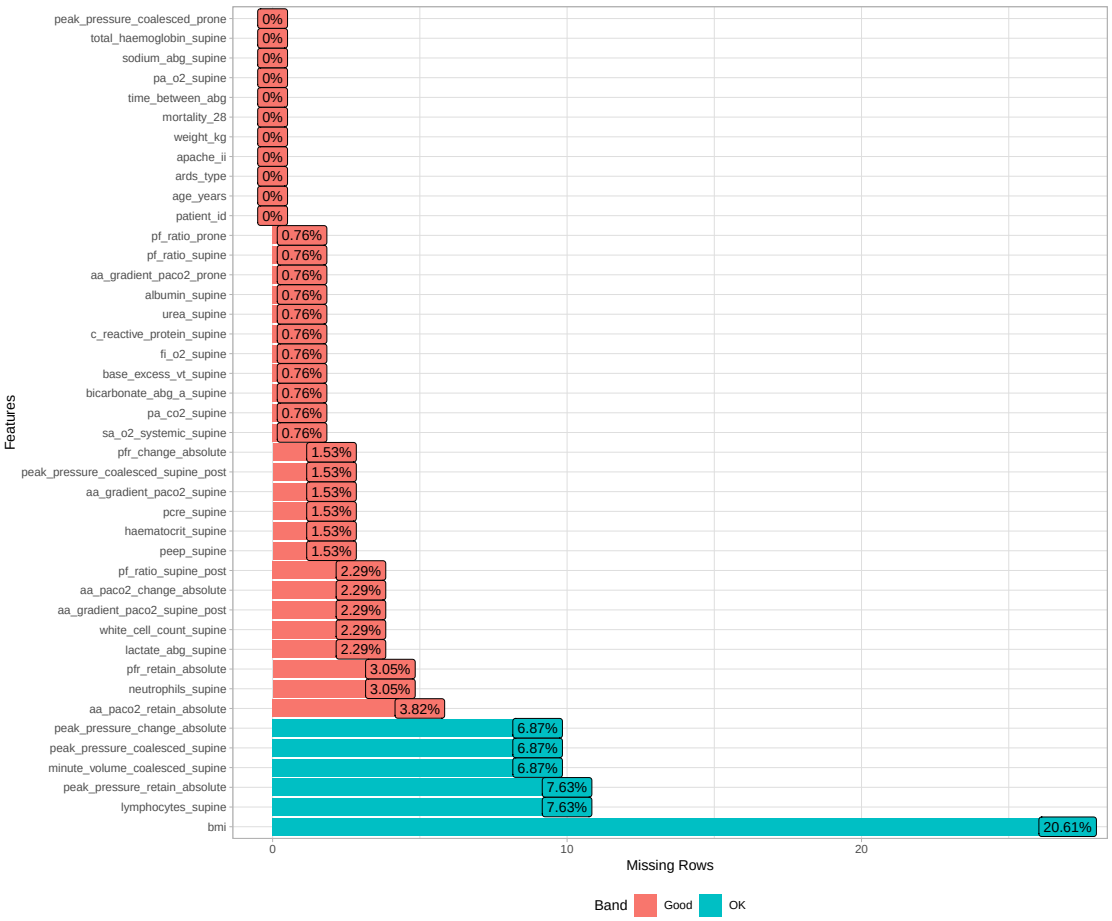

Figure 1: Variables and percentage of missing values

Demographics

Table 2 shows the demographics of the patients who were sampled. They have been subdivided into broad categories describing their lung injury, as determined from the electronic notes, and classified as per the following table.

Table 2: Demographics of sampled patients

Characteristic	**N = 133**[1]
Gender	
f	45 (34%)
m	88 (66%)
Age (years)	58 (15)
Height (cm)	169 (13)
Not recorded	27
Weight (kg)	86 (21)
ARDS category	
ARDSp	73 (55%)
Covid_19	51 (38%)
ARDSexp	5 (3.8%)
Unknown	4 (3.0%)
Admitting location	
cath lab	1 (0.8%)
ccu	3 (2.3%)
ed	27 (20%)
guh_ward	58 (44%)
theatre (elective)	3 (2.3%)
theatre (emergency)	4 (3.0%)
transfer	37 (28%)
Length of stay (days)	19 (18)
Apache II score	19 (8)
Proning sessions	3 (2)
Outcome	
dc	72 (54%)
rip	61 (46%)
BMI	31 (8)
Not recorded	27

[1] n (%); Mean (SD)

Predicting 28-day mortality

General approach

The work was conducted using R 4.2.2 in RStudio version 2023.06.0+421. Modelling was performed mostly through the `tidymodels` suite of packages. Five models were trained:

1. Logistic Regression
2. Gaussian Naive Bayes
3. C5.0 boosted tree
4. XGBoost
5. Support Vector Machine

For each of these models, the data was sampled using k-fold cross validation (k=5). The folds were stratified by 28-day mortality to ensure that each fold was balanced in an approximately similar way with regards to the outcome variable.

Each model was fit to the 80% training split and predictions made on the 20% testing split of each fold.

Outcome variable and measure of performance

The outcome variable is 28-day mortality following the initial session of prone positioning i.e. is the patient alive or dead 28 days after their initial proning session. Of a total of 131 patients observed, 57 were dead within 28 days of their initial session of prone positioning, and 74 were alive.

Individual models were compared using the F1 score. This is calculated as shown:

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Data folds and splits

To prepare the data for use in the each model, k-fold cross-validation was performed. This was performed for k = 5, and with 5 repetitions. Within each fold, the training data was then split into two partitions, with 80% of data used for training and 20% used for testing the fitted model. This was accomplished using the `rsamples` package.

```
data_v_fold <-
  vfold_cv(data = prone_session_1,
           v = 5,
           repeats = 5,
           strata = mortality_28)
```

Logistic Regression

Logistic regression was performed by training a model using the `glmnet` engine with regularization. For this, the `parnsip` package was utilized. The regularization type was Lasso, and the hyperparameter for amount of penalization/regularization was tuned using the `tune` package with a maximum entropy grid search with 10 values.

```
lr_model_01 <-
  logistic_reg(mode = 'classification',
               engine = 'glmnet',
               penalty = tune(),
               mixture = 1)

lr_tuning_grid <-
  grid_max_entropy(penalty(range = c(-5, 3)),
                   size = 10)
```

Before data was fed to this, it was prepared by:

- removing variables that had >5% of values missing
- one-hot encoding all factor variables,
- imputing any remaining missing values using bagged trees
- removing parameters if it and any other have a Pearson correlation coefficient > 0.8
- removing any parameters with zero variance

```
lr_recipe <-
  recipe(prone_session_1, formula = mortality_28 ~ .) %>%
  step_rm(patient_id,
          bmi,
          weight_kg) %>%
  step_dummy(all_factor_predictors(), -mortality_28) %>%
  step_impute_bag(all_predictors()) %>%
  step_corr(all_numeric_predictors(), threshold = 0.8) %>%
  step_zv()
```

The model was trained, tuned, and tested on each fold. The best performing penalty term is shown below, and is the result of an average value taken for all the models in each fold. The five best performing terms are shown below.

L1 Penalty	Metric	Mean Value	Standard Error
0.0841	F1	0.752	0.008
0.0117	F1	0.729	0.014
0.463	F1	0.722	0.002
4.96	F1	0.722	0.002
50.6	F1	0.722	0.002

Table 3: Five best logistic regression models with L1 penalty terms, by F1 score

C5.0 Boosted Tree

A boosted tree model was developed using the `C5.0` engine. The three hyperparameters to be tuned were-

- `trees`: the number of trees within each model
- `min_n`: minimal size of each node
- `sample_prop`: the proportion of observations used in each sample

```
c5Boost_mod <-
  boost_tree(mode = 'classification',
             engine = 'C5.0',
             trees = tune(),
             min_n = tune(),
             sample_size = tune()
             ) %>%
  set_args(earlyStopping = FALSE)
```

These hyperparameters were tuned using a maximum entropy grid.

```
c5Boost_grid <-
  grid_max_entropy(trees(range = c(1, 10)),
                   min_n(),
                   sample_prop(range = c(0.1, 0.999)),
                   size = 100,
                   variogram_range = 0.5
                   )
```

Preparation of the data prior to being used in the model involved:

- removal of `patient_id`
- removal of `bmi` (~20% missing values)
- removal of `weight_kg` (due to concerns about accuracy)
- removal of zero variance predictors
- removal of parameters if it and any other have a Pearson correlation coefficient > 0.8

```
c5Boost_recipe <-
  recipe(prone_session_1, formula = mortality_28 ~ .,) %>%
  step_rm(patient_id,
          bmi,
          weight_kg) %>%
  step_zv(all_predictors()) %>%
  step_corr(all_numeric_predictors() , threshold = 0.8)
```

The model was trained, tuned, and tested on each fold. The best performing penalty term is shown below, and is the result of an average value taken for all the models in each fold. The five best performing terms are shown below.

Trees	Minimum n	Sample Proportion	Metric	Mean Value	Standard Error
10	24	0.96	F1	0.729	0.016
7	17	0.83	F1	0.727	0.016
4	13	0.20	F1	0.724	0.002
3	25	0.15	F1	0.723	0.002
9	29	0.74	F1	0.723	0.012

Table 4: Ranking of five best performing C5.0 hyperparameter combinations by F1 score

Gaussian Naive Bayes

The Gaussian Naive Bayes classifier is a probabilistic approach to classification and uses only continuous variables. There are no hyperparameters to tune for this model, so data preparation is key.

- variables that had >10% missing values, and variables with sparse distribution and strong skews were removed (`bmi`, `fi_o2_supine`, `minute_volume_coalesced_supine`, `peak_pressure_coalesced_supine`, `peak_pressure_coalesced_prone`, `peak_pressure_coalesced_supine_post`)
- one-hot encoding of factor variables
- removal of parameters if it and any other have a Pearson correlation coefficient > 0.8
- Yeo-Johnson transformation performed on all numeric variables
- normalisation of all numeric variables

This model is facilitated by the `naivebayes` package, rather than the `tidymodels` metapackage. There is no need to specify or configure the model. There are no hyperparameters to tune.

```
# initiate empty df for performance
gnb_performance <- tibble(fold = 0,
                          sensitivity = 0,
                          specificity = 0,
                          ppv = 0,
                          npv = 0,
                          f_measure = 0,
                          kappa = 0)

# create fold object
folds <- 25
gnb_folds <-
  prone_session_1 %>%
  vfold_cv(v = sqrt(folds),
           repeats = sqrt(folds),
           strata = mortality_28)


# loop to train, test, create measures of gnb models
for (i in 1:folds) {
  fold_train <- gnb_folds$splits[[i]] %>% analysis()
```

```
fold_train <- bake(object = gnb_recipe,
                   new_data = fold_train,
                   composition = 'tibble')

fold_train_predictors <-
  fold_train %>%
  select(-mortality_28) %>%
  as.matrix()

fold_train_outcome <- fold_train$mortality_28

gnb_model <- gaussian_naive_bayes(x = fold_train_predictors,
                                  y = fold_train_outcome)

fold_test <- gnb_folds$splits[[i]] %>% assessment()

fold_test <- bake(object = gnb_recipe,
                  new_data = fold_test,
                  composition = 'tibble')

fold_test_predictors <-
  fold_test %>%
  select(-mortality_28) %>%
  as.matrix()

fold_test_outcome <- fold_test$mortality_28

fold_test$pred_outcome <-
  predict(object = gnb_model,
          newdata = fold_test_predictors, type = 'class')

fold_test$pred_prob <-
  predict(object = gnb_model,
          newdata = fold_test_predictors,
          type = 'prob')[, 1]

# metrics
sens_test <- sensitivity(data = fold_test,
                         truth = mortality_28,
                         estimate = pred_outcome)
spec_test <-specificity(data = fold_test,
                        truth = mortality_28,
                        estimate = pred_outcome)
ppv_test <- ppv(data = fold_test,
                truth = mortality_28,
                estimate = pred_outcome)
```

```
  npv_test <- npv(data = fold_test,
                  truth = mortality_28,
                  estimate = pred_outcome)
  fmeas_test <- f_meas(data = fold_test,
                       truth = mortality_28,
                       estimate = pred_outcome)
  kappa_test <- kap(data = fold_test,
                    truth = mortality_28,
                    estimate = pred_outcome)

  gnb_performance <- add_row(gnb_performance,
                             fold = i,
                             sensitivity = round(sens_test$.estimate, 2),
                             specificity = round(spec_test$.estimate, 2),
                             f_measure = round(fmeas_test$.estimate, 2),
                             kappa = round(kappa_test$.estimate, 2))

}

# tidy the accuracy params tibble
gnb_performance <- gnb_performance[2:(folds + 1), ]

gnb_performance <-
  gnb_performance %>%
  summarise(sens_mean = round(mean(sensitivity), 2),
            spec_mean = round(mean(specificity), 2),
            f1_mean = round(mean(f_measure), 2),
            kappa_mean = round(mean(kappa), 2))
```

The classifier was trained and fit to all five folds and repeats. Mean results are shown below:

Sensitivity	Specificity	F1 score	Kappa
0.81	0.53	0.75	0.35

Table 5: Mean values of performance parameters for Gaussian Naive Bayes classifier

Other approaches

In addition to the above approaches, XGBoost was used, a was a Support Vector Machine approach. The details have not been included here, but these models performed poorly and did not better than chance despite extensive tuning.

Next steps

There is a consistent pattern that emerges from any modelling approaches that perform even moderately well. Overall performance is decent but not spectacular, sensitivity is quite good but specificity is not very much better than chance.

The only possibility I see to try and squeeze a little extra would be a simple ensemble approach where I would employ a majority voting approach. Given the limitations on the size of my data, I have considered making another random fold from the data and fitting and predicting using it. However, I also feel at this point that I have tried to work this (admittedly small and problematic) data fairly well.

I would really love this work to lead to some sort of publication given how hard I have tried and how much Machine Learning knowledge I have acquired. I am interested in your thoughts.

References

1. Guérin C, Reignier J, Richard JC, et al. Prone positioning in severe acute respiratory distress syndrome. *The New England journal of medicine*. 2013;368(23):2159-2168. doi:10.1056/NEJMoa1214103
2. Banavasi H, Nguyen P, Osman H, Soubani AO. Management of ARDS–what works and what does not. *The American Journal of the Medical Sciences*. 2021;362(1):1323.
3. Chen W, Ware LB. Prognostic factors in the acute respiratory distress syndrome. *Clinical and Translational Medicine*. 2015;4:23. doi:10.1186/s40169-015-0065-2