Survival Analysis on Li-ion Batteries

Using R version 4.5.0

Matias BOTTARINI

Introduction

For this project, I wanted to find a data set that applies survival analysis to renewable energies. I found a paper which uses survival analysis to predict the EOL of Toyota battery cells and which data set is openly available.
The aim of this document is to reproduce some of the results in [1] and generate several models to predict the lifetime of a cell after collecting its voltage curves. Battery degradation is directly related to the internal resistance, the voltage and capacity of the batteries. The method proposed in [1] uses the voltage charge and discharge curves on the first 50 cycles as covariates. The voltage curves will change their form as the batteries degrade. Moreover, using the voltage as feature eases implementation, as it is monitored by the BMS and it could be easily retrieved, thus showing the practical approach of the method.

I focused on studying the models using the voltage charge curves' variables. The proposed models in this report are :

1. **Cox Proportional Hazard**
2. **AFT Weibull**
3. **Survival Random Forest**

I will compare the results obtained with these models to the results from the **XGBoost implementation of the CoxPH model** used in the paper.

ETL

From the data source I used the data treatment scripts in python and translated them into R to have the same data sets as in [1].

The raw data set has a size of 2.6 GB and contains the internal resistance, voltage, capacity (Qr and Qd) and temperatures measured during the deep cycling of 8 battery batches with 46-48 cells (373 cells in total). **From this initial 373 cells, 362 are left** after the data treatment.

During the data treatment I collected the main features, the time-to-event (T) and the indicator (δ) as well as the voltage charge and discharge features used in [1] for training the models. The internal resistance, temperature and capacity data where omitted. I processed the raw data into two data sets:

1) **toyota_structured_data.rds** which contains the voltage values on the charging/discharging cycles for each cell. Available here.
2) **data_treated.rds** which contains the voltage features (signature paths) on charge and discharge with labels *ch_{feature_number}* and *dch_{feature_number}* along with the EOL time (T) and the censor indicator (δ) for each cell. Available here.

I modified the scripts to import as well the features used in the paper. These features are the path signatures of the voltage curves. By using path signatures on the charge and discharge voltage curves, we can compress the information of these curves into a few features. Particularly, when using a signature of the form $Sig_i(t) = (t, V(t))$ with a signature depth $k = 2$, we obtain **6 features for each cycle**. Nonetheless, the author of the paper wants to express it in a way where we compress the voltage of the first 50 cycles for each cell. Thus, when recalculating the paths of the form $P_i = (cycle_i, Sig_i(t))$ for the first 50 cycles and with signature depth of $k = 2$, we obtain 36 features (6^2) for each cell.

Therefore, we observe 36 features for each cell on charging and 36 features for discharging.

```r
suppressPackageStartupMessages({
    library(tidyverse)
library(survival)
library(survivalROC)
library(gtsummary)
library(broom)
library(ggsurvfit)
library(ggrepel)
})

df = readRDS("toyota_structured_data.rds")
data = as.data.frame(readRDS(file="data_treated.rds"))
```

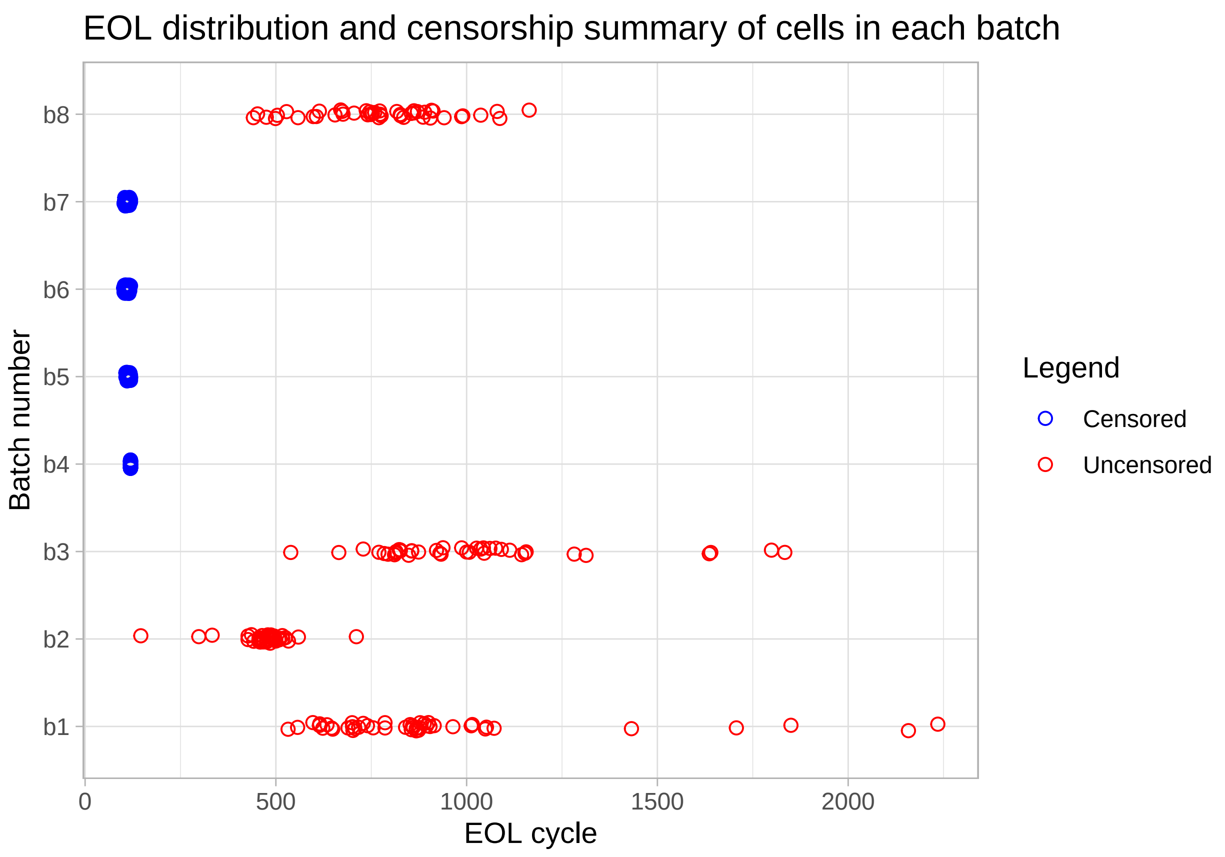
EDA

For a fully charged cell, the voltage is around 3.5 V and the full capacity 1.1 Ah. The EOL (End Of Life) of the cell is estimated when the capacity has fallen down below 80% of the batteries capacity, i.e. 0.88 Ah. The EDA plots could be found in the Annex, see github repository.

The experiments were done with different charging and discharging policies, so we can assume a random behavior when studying the survival risk.

From the 8 batches, 4 were driven to the EOL (b1, b2, b3 and b8) and 4 were censored i.e. they left the experiment without arriving to EOL (b4, b5, b6 and b7). See the following figure (see Figure 3 in [1]).

```r
BATCHES = c("b1","b2","b3","b4","b5","b6","b7","b8")
x_batches = c()
y_eol = c()
censorship = c()
tmp_batch = c()
for (label in BATCHES) {
  for (i in 1:length(df)) {
    if (df[[i]][["summary_data"]][["batch_name"]] == label)
      {
      x_batches = rbind(x_batches, label)
      y_eol = rbind(y_eol,df[[i]][["summary_data"]][["end_of_life"]])
        if (label %in% c("b4","b5","b6","b7"))
          {censorship = rbind(censorship, "Censored") }
        else {censorship = rbind(censorship, "Uncensored")       }
      }
  }
}
df_eol = data.frame(batches=x_batches, eol=y_eol, censorship=censorship)
ggplot(data = df_eol, aes(x=eol, y=batches)) +
  geom_jitter(height=0.05, width = 0, size = 2, shape = 1, aes(colour = censorship)) +
  labs(x = "EOL cycle",
       y = "Batch number",
       color = "Legend",
       title = "EOL distribution and censorship summary of cells in each batch") +
    scale_color_manual(values = c("blue","red")) +
  theme_light()
```



We observe that the censored batches never pass the 200 cycles and that most of the EOL happens between 500 and 1000 cycles. The mean is 440 cycles and the median is 119 cycles.

```
summary(data$time_to_failure)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   101.0   113.0   119.0   439.9   754.5  2235.0
```

EDA: survival analysis

We have a total of 174 batteries arriving to EOL, with a median survival time of 771 cycles.

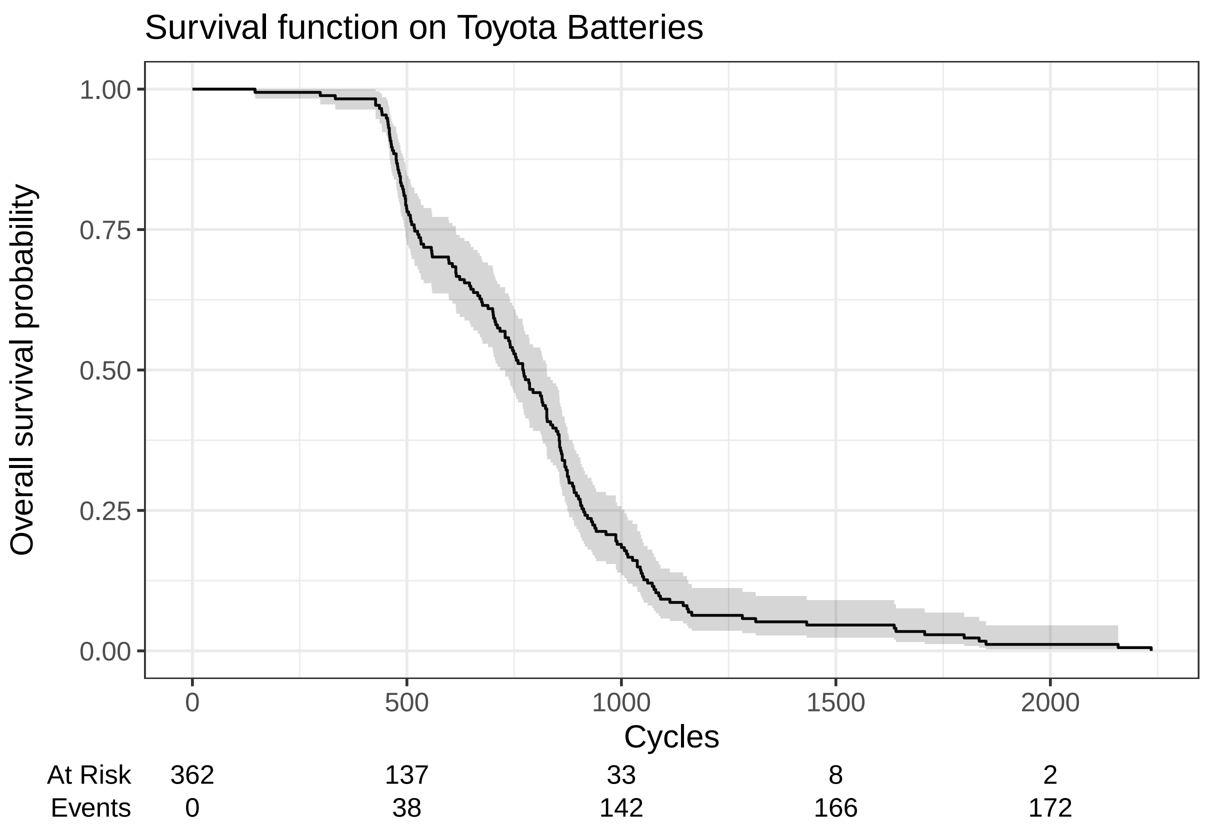
```
fit.KM <- survfit(Surv(time_to_failure, failure) ~ 1, data = data)
print(fit.KM)
```

```
## Call: survfit(formula = Surv(time_to_failure, failure) ~ 1, data = data)
##
##         n events median 0.95LCL 0.95UCL
## [1,] 362    174    771     717     826
```

```
fit.ggKM <- survfit2(Surv(time_to_failure, failure) ~ 1, data = data)
```

Next, we plot the survival curve of the Toyota batteries. We could also plot the CFD function, but to reduce space we are going to omit it. The CFD could be plotted by giving the argument *type* = *"risk"*.

```
  ggsurvfit(fit.ggKM, type = "survival") +
  labs(
    x = "Cycles",
    y = "Overall survival probability",
    title = "Survival function on Toyota Batteries"
    ) +
  add_confidence_interval() +
  add_risktable()
```



Method

To be able to compare the results w.r.t. the paper, we need to follow the same methods.

In terms of data treatment, we will split the data and apply a scale on it based on the Inter Quantile Range (RobustScaler in ScikitLearn).

Regarding the results comparison, we are going to compare each model by the **C-index** (concordance index) and the **integrated Brier Score (IBS)** as well as the **integrated Cumulative Dynamic AUC** for $t \in [500, 1000]$ *cycles*.

Split data into a training and a testing set

```
set.seed(42)
test_split = 0.2
n_obs = length(data$failure)
i.training <- sample.int(n_obs, size = ceiling(n_obs*(1-test_split)), replace = FALSE)
```

```
i.testing <- setdiff(seq_len(n_obs), i.training)
d_train <- data[i.training,]
d_test <- data[i.testing,]
```

Cox Proportional model

We are going to start with the full model to later make a variable selection with two methods, one with the step function (AIC criterion) and another with the penalized method (elasticNet method).

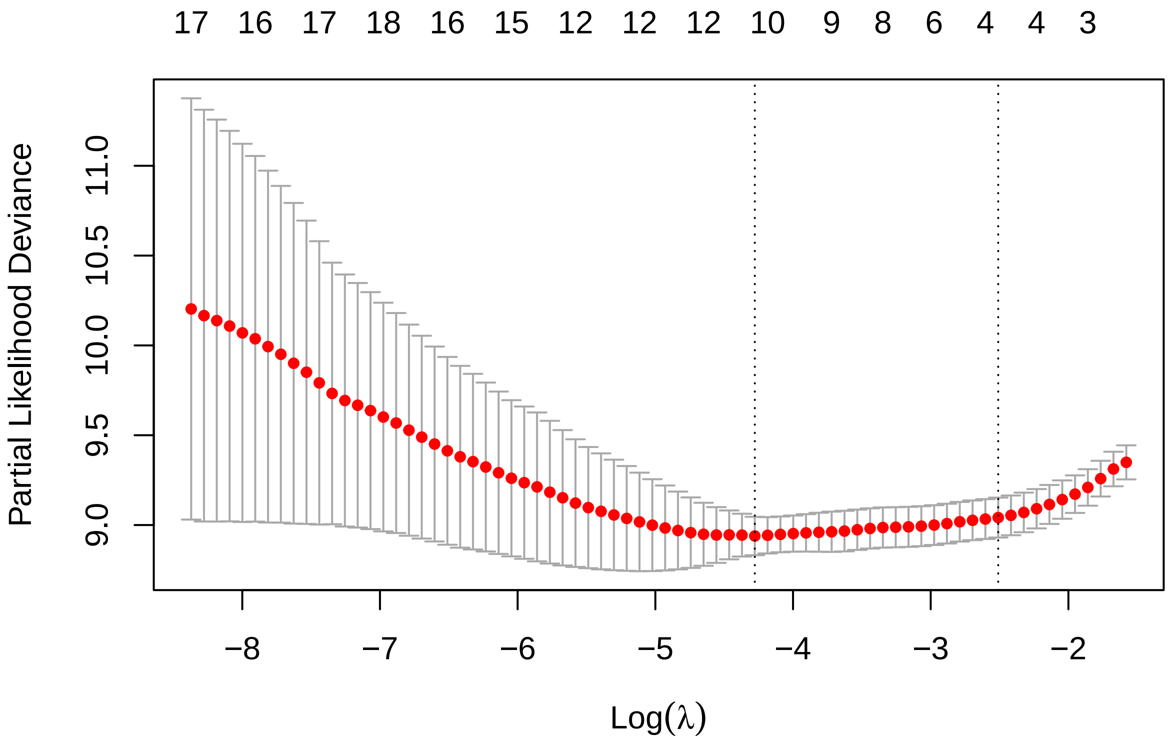
```
CPH_ch_full <- coxph(Surv(time_to_failure, failure) ~ ch_1+ch_2+ch_3+ch_4+ch_5+ch_6+ch_7+ch_8+ch_9+ch_1
```

The variables selected by the step function are: V2, V6, V10, V12, V14, V18, V22, V24, V26, V28, V30, V32 and V34.

```
CPH_ch_step <- step(CPH_ch_full, trace = F)
```

We have balanced the penalized regression with $\alpha = 0.9$ to introduce a non-zero Ridge term (0.1) with respect to the Lasso term (0.9) on the ElasticNet cost function.

```
suppressPackageStartupMessages(library(glmnet))
y_train = Surv(d_train$time_to_failure, d_train$failure)
X_train_ch = as.matrix(d_train[,-c(1,2,3,40:76)])
X_train_dch = as.matrix(d_train[,-c(1,2,3,4:39)])
CPH_ch_pen <- glmnet(X_train_ch, y_train, family = "cox", alpha = 0.9)
CPH_ch_pen.cv10 <- cv.glmnet(X_train_ch, y_train, family = "cox")
plot(CPH_ch_pen.cv10)
```



We are going to select the higher number of variables selected, which corresponds to λ_{min}. We can observe that the variables selected by the penalized method are different from the AIC's step method. The selected variables for a $\lambda_{min} = 0.01388$ are V4,V5,V6,V10,V18,V22,V23,V24,V30,V32 and V35.

```r
#coef(CPH_ch_pen.cv10, s = "lambda.min")
#CPH_ch_pen.cv10$lambda.min
CPH_ch_pen_lamdbda_min <- coxph(formula = Surv(time_to_failure, failure) ~ ch_4 + ch_5 +
    ch_6 + ch_10 + ch_18 + ch_22 + ch_23 + ch_24 + ch_30 + ch_32 +
    ch_35, data = d_train, model = T, x = T, y=T)
```

```r
models <- list(
    CPH_ch_Full = CPH_ch_full,
    CPH_ch_step = CPH_ch_step,
    CPH_ch_pen = CPH_ch_pen_lamdbda_min
  )
map_dbl(models, ~ summary(.)$concordance[1])
```

```
## CPH_ch_Full CPH_ch_step  CPH_ch_pen
##   0.7457203   0.7537578   0.7553236
```

```r
sapply(models, AIC)
```

```
## CPH_ch_Full CPH_ch_step  CPH_ch_pen
##    1016.236    1008.864    1012.299
```

Without using the testing data set yet, the penalized model has a higher concordance index. Logically, the step model has higher AIC score.

```r
cph_ch.mod <- CPH_ch_pen_lamdbda_min
```

```r
y_test = Surv(d_test$time_to_failure, d_test$failure)
X_test_ch = d_test[,-c(1,2,3,40:76)]
X_test_dch = d_test[,-c(1,2,3,4:39)]

# Select the variables from the penalized model
X_test_ch_sel = X_test_ch[,c(4,5,6,10,18,22,23,24,30,32,35)]
```

I used the *survex* library (R version 4.5.0) to calculate the Integrated Brier Score and The Integrated Cumulative Dynamic AUC between 500 and 1000 cycles.

```r
suppressPackageStartupMessages(library(survex))
cph_explainer <- survex::explain(cph_ch.mod, verbose = F)
score_times = seq(500,1000,10)
surv_pred <- cph_explainer$predict_survival_function(model = cph_ch.mod,
                                                     newdata = X_test_ch_sel,
                                                     times = score_times)
risk <- cph_explainer$predict_function(model = cph_explainer$model,
                                       newdata = cph_explainer$data)
cph_ch.mod$C_idx <- c_index(y_true = cph_explainer$y, risk = risk)
cph_ch.mod$IBS = integrated_brier_score(y_test, surv = surv_pred, times = score_times)
cph_ch.mod$ICD_AUC =  integrated_cd_auc(y_test, surv = surv_pred, times = score_times)
cph_ch.mod$AIC <- AIC(cph_ch.mod)
```

I compared the step and the penalized models, and we have obtained better scores (higher C-index and AUC and lower IBS) for **the penalized selected CPH model**. To save space, we are not going to show this comparison.

```r
cph_ch.mod$sch_residual <- cox.zph(cph_ch.mod)
cph_ch.mod$sch_residual
```

```
##          chisq df     p
## ch_4    0.3351  1 0.563
## ch_5    5.8575  1 0.016
## ch_6    2.2880  1 0.130
## ch_10   2.7902  1 0.095
## ch_18   3.2567  1 0.071
## ch_22   1.7803  1 0.182
## ch_23   5.7751  1 0.016
## ch_24   0.0154  1 0.901
## ch_30   4.6380  1 0.031
## ch_32   0.9548  1 0.329
## ch_35   2.7863  1 0.095
## GLOBAL 22.2864 11 0.022
```

By checking the Schoenfeld global residuals we CAN NOT assume that the hazard rates are proportional for a confidence level of 0.05. Therefore, we could think about using stratification or truncation; reduce the confidence level to < 0.02 and accept the H0 or try to catch the non-proportionality with an AFT model.

AFT parametric model

Because we don't have proportionality of hazards on the CPH model, I decided to train an AFT model and calculate the scores. The object *aft_ch.pred* has the predictions on the test set and has a dimension of $dim = (\ 72, 51\ )$. The score times has 51 time points (from 500 to 1000) and we have 72 voltage charge curve observations in the test set.

We find the best score with parameters *dist="weibull"* and an Accelerated Failure Time parameter *param="lifeAcc"*. All the possible parameters in the *ldatools* package where tested. The fitting of the shape converges, and thus we can use this model if needed.

```r
suppressPackageStartupMessages({
  library(eha) # for using the "aftreg" function
  library(ldatools)
  library(SurvMetrics)})
aft_ch.mod <- aftreg(formula = Surv(time_to_failure, failure) ~ ch_4 + ch_5 + ch_6 + ch_10 + ch_18 + ch
                             data = d_train,
                             dist = "weibull",
                             param = "lifeAcc",
                             model = T, x = T, y=T)
cat("AFT model fitting optimization converged:", aft_ch.mod$convergence)
```

```
## AFT model fitting optimization converged: TRUE
```

```r
new_data = data.frame(X_test_ch_sel,time_to_failure=d_test$time_to_failure,failure=d_test$failure)

# from ldatools library, we calculate the survival curves
aft_ch.pred <- predictSurvProb(aft_ch.mod, newdata = new_data, times = score_times)
# from SurvMetrics we use the Cindex function on t c [500, 1000]
c_indexes <- c()
for (t_idx in 1:dim(aft_ch.pred)[2]) {
  C_idx <- Cindex(object = y_test, predicted = aft_ch.pred[,t_idx], t_star = score_times[t_idx])
  c_indexes <- append(c_indexes, C_idx)
}
aft_ch.mod$C_idx <- mean(c_indexes)
aft_ch.mod$IBS <- integrated_brier_score(y_test, surv = aft_ch.pred, times = score_times)
aft_ch.mod$ICD_AUC <-  integrated_cd_auc(y_test, surv = aft_ch.pred, times = score_times)
aft_ch.mod$AIC <- AIC(aft_ch.mod)
```

Survival Random Forest Model

First, we started training a Survival Random Forest model with all the charge voltage variables, giving us the following table:

```
srf_full_ch <- c(C_idx=0.8255741, IBS=0.08753846, IDC_AUC=0.7698247)
srf_full_ch
```

```
##      C_idx        IBS    IDC_AUC
## 0.82557410 0.08753846 0.76982470
```

However, as we want to compare the previous models, we are going to select the same variables as the penalized Cox PH model, so we can compare the models on the basis of identical variables.

The combination of parameters *splitrule="logrank"* and *ntree=500* gave the best scores. On the other hand, I tried to use the variable importance function of the Random Forest (see annex in github code) to make a variable selection, but the model is not better than the one proposed here.

```
suppressPackageStartupMessages(library(randomForestSRC))
srf_ch.mod <- rfsrc(Surv(time_to_failure, failure) ~
                      ch_4 + ch_5 + ch_6 + ch_10 + ch_18 + ch_22 + ch_23 + ch_24 + ch_30 + ch_32 + ch_35
                    data=d_train,
                    splitrule = "logrank",
                    ntree = 500)

srf.explainer <- survex::explain(srf_ch.mod, verbose = F)

surv_pred <- srf.explainer$predict_survival_function(model = srf.explainer$model,
                                                     newdata = X_test_ch_sel,
                                                     times = score_times)
risk <- srf.explainer$predict_function(model = srf.explainer$model,
                                       newdata = srf.explainer$data)

srf_ch.mod$C_idx <- c_index(y_true = srf.explainer$y, risk = risk)
srf_ch.mod$IBS = integrated_brier_score(y_test, surv = surv_pred, times = score_times)
srf_ch.mod$ICD_AUC =  integrated_cd_auc(y_test, surv = surv_pred, times = score_times)
srf_ch.mod$AIC <- NA
```

Results and discussion

```
score_table <- c(cph_ch.mod$C_idx, cph_ch.mod$IBS, cph_ch.mod$ICD_AUC, cph_ch.mod$AIC)
score_table <- rbind(score_table,
                     c(aft_ch.mod$C_idx, aft_ch.mod$IBS, aft_ch.mod$ICD_AUC, aft_ch.mod$AIC))
score_table <- rbind(score_table,
                     c(srf_ch.mod$C_idx, srf_ch.mod$IBS, srf_ch.mod$ICD_AUC, srf_ch.mod$AIC))
score_table <- rbind(score_table, c(0.794, 0.112, 0.9015, NA))
row.names(score_table) <- c("CPH", "AFT weibull", "Survival Random Forest", "XGBoost (Paper mean values)
colnames(score_table) <- c("C-idx","IBS","ICD_AUC","AIC")
score_table
```

```
##                                 C-idx        IBS   ICD_AUC      AIC
## CPH                         0.7553236 0.13803045 0.4630719 1012.299
## AFT weibull                 0.7529410 0.14532658 0.4659791 1876.491
## Survival Random Forest      0.8249478 0.08738262 0.7755177       NA
## XGBoost (Paper mean values) 0.7940000 0.11200000 0.9015000       NA
```

As we can observe in the table, the models CPH (non-parametric) and AFT weibull (parametric) perform similarly. However, the AIC on the CPH model is much lower than on the AFT model. As the CPH model did not pass the proportional hazard test, we would recommend choosing the AFT model over the CPH.

On the other hand, looking at the SRF (Survival Random Forest) model, we see that outperforms the rest of the models and even the results published in the paper **for the C-index and the IBS scores**.

The most significant discrepancies are shown in the AUC score, which is at its lowest with the CPH and AFT models, while the SRF is closer to the XGBoost.

We can conclude that, due to its good performance on every score, the model in the paper is more robust than those proposed in this document.

Conclusion

For simplification and to be able to compare models, we have respected the variable selection done at the beginning of the **Method chapter**. However, in the original paper [1], all the voltage charge variables are used for training an XGBoost model. In this document, a Survival Random Forest model is also trained using all the charge voltage variables, giving better scores for the C-index and the IBS (for one test set) than those shown in the paper (mean value on 100 test sets).

We have observed that a parametric model such as AFT has its limitations as it depends on the distribution. The weibull distribution was good enough to fit the data and give similar results as the Cox PH model.

On the other hand, if we use a Cox PH and we **do not have proportional hazards** (our case), we could truncate or stratify the data. Stratification was limiting as we could not easily classify the path signatures of the voltage curves (*strat*). Regarding truncation, we could think about truncating the data between 300 and 1200 cycles, where most of the event occur, and see if we pass a proportional hazard test, but with the drawback of loosing explainability on the whole lifecycle.

To solve this a Survival Random Forest was proposed and gave the best results amid the other proposed models.

In conclusion, we have reproduced the methodology in [1] and obtained very good results with our Survival Random Forest model for the charging voltage curve model. Other models were tested, but since Survival Random Forest outperforms the rest and its more flexible in terms of dealing with non-linearities, we could recommend it to solve this topic.

NB: The same methodology can be followed to produce models based on the **discharge voltage curves**.

Code availability

The experiments carried out on this R markdown document can be found in https://github.com/zBotta/surv An-batteries. The objective of this code was purely academic in order to address a course project on Survival Analysis.

Bibliography

[1] Rasheed Ibraheem et al., Robust survival model for the prediction of Li-ion battery lifetime reliability and risk functions
[2] David Kuhajda, Using Survival Analysis to Evaluate Medical Equipment Battery Life
[3] Rasheed Ibraheem et al., Early prediction of Lithium-ion cell degradation trajectories using signatures of voltage curves up to 4-minute sub-sampling rates
[4] Attia Peter M. et al., Statistical Learning for Accurate and Interpretable Battery Lifetime Prediction
[5] Attia Peter M. et al., Data-driven prediction of battery cycle life before capacity degradation

Annex

EDA: Extra plots

In the figure below, we observe the evolution of the capacity of cell 45 from batch 1 (b1c45). The capacity decreases as it has been cycled, until arriving to its EOL at 597 cycles.

```
capacity_b1c45 = df[["b1c45"]][["summary_data"]][["discharge_capacity"]]
cycles_b1c45 = df[["b1c45"]][["summary_data"]][["cycle"]]
df_b1c45 = data.frame(capacity=capacity_b1c45, cycles=cycles_b1c45)
EOL_b1c45 = df[["b1c45"]][["summary_data"]][["end_of_life"]]
df_eol = data.frame(eol_cycle=EOL_b1c45, eol_capacity=capacity_b1c45[EOL_b1c45])

jpeg('capacity_loss_until_eol.jpeg')
ggplot(data = df_b1c45, aes(x = cycles_b1c45, y = capacity_b1c45)) +
  geom_line(lwd=1) +
  labs(x = "Cycles",
       y = "Capacity (Ah)") +
  geom_point(data = df_eol,
             aes(eol_cycle,eol_capacity), size = 3, col="red") +
  geom_label_repel(data          = df_eol,
                   aes(eol_cycle, eol_capacity),
                   label="EOL",
                   size          = 4,
                   box.padding   = 1.5,
                   point.padding = 0.5,
                   force         = 100,
                   segment.color = "red",
                   direction     = "x")
dev.off()
```

```
## pdf
##   2
```

Below the charge and discharge voltage curves of cell 45 form the batch 1 (b1c45) at cycle 50. Plotting all curves becomes difficult as the time stamp is different for each charge-discharge experiment. During the data treatment, data has been interpolated to respect a constant time stamp.

```
t_ch_b1c45 = df[["b1c45"]][["cycle_data"]][["50"]][["charge"]][[1]]
V_ch_b1c45 = df[["b1c45"]][["cycle_data"]][["50"]][["charge"]][[2]]
df_ch_b1c45 = data.frame(t=t_ch_b1c45, V=V_ch_b1c45)
t_dch_b1c45 = df[["b1c45"]][["cycle_data"]][["50"]][["discharge"]][[1]]
V_dch_b1c45 = df[["b1c45"]][["cycle_data"]][["50"]][["discharge"]][[2]]
df_dch_b1c45 = data.frame(t=t_dch_b1c45, V=V_dch_b1c45)
p1 <- ggplot(data = df_ch_b1c45, aes(x = t, y = V)) +
  geom_line(lwd=1, colour="red") +
  labs(x="Time (min)",
       y = "Voltage (V)",
       title = "Charging and Discharging curves for b1c45 at cycle 50") +
  theme_light()
p2 <- ggplot(data = df_dch_b1c45, aes(x = t, y = V)) +
  geom_line(lwd=1, colour="green")  +
  labs(x="Time (min)",
       y = "Voltage (V)") +
  theme_light()
library(patchwork )
```

Characteristic	70% Percentile
Overall	878 (856, 933)

```r
jpeg('voltage_curves_on_cell-b1c45.jpeg')
p1 + p2
dev.off()
```

```
## pdf
##   2
```

Use cases of battery survival curve

In a real world scenario applied on batteries, the survival curves could be used for:
1) Estimating the amount of batteries that a battery producer should have in stock to replace their installed battery packs. For instance, if we have a contract with our clients stating that we had to change the batteries when at least 70% of the batch cells arrive to its EOL, we could estimate after how many cycles we would need to start having a new battery on stock.

The batteries experiencing at least 878 cycles with a 95%CI, would need to be replaced.

```r
library(gtsummary )
tbl_survfit(fit.KM , probs = c(0.7))
```

2) As seen in paper [2], we could estimate the maintenance shifts of a given battery fleet to ensure a certain capacity (survival rate).

Variable selection with SRF

```r
# RF variable selection
srf_ch.mod <- holdout.vimp(Surv(time_to_failure, failure) ~ ch_1+ch_2+ch_3+ch_4+ch_5+ch_6+ch_7+ch_8+ch_

# taking 1 sd
one_sd_vimp = sd(na.omit(srf_ch.mod$importance))
abs(na.omit(srf_ch.mod$importance)) > one_sd_vimp
```

```
##  ch_1  ch_2  ch_3  ch_5  ch_6  ch_7  ch_8  ch_9 ch_10 ch_11 ch_12 ch_13 ch_14
## FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
## ch_16 ch_17 ch_18 ch_19 ch_20 ch_21 ch_23 ch_24 ch_25 ch_26 ch_27 ch_28 ch_30
##  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## ch_31 ch_33 ch_34 ch_35
##  TRUE FALSE FALSE FALSE
```

The selected variables are V2, V3, V4, V6, V7, V8, V11, V15, V21, V23, V25, V35, V36.

```r
srf_ch_vimp.mod <- rfsrc(Surv(time_to_failure, failure) ~  ch_2 + ch_3 + ch_4 + ch_6 + ch_7 + ch_8 + ch

X_test_ch_vimp = X_test_ch[,c(2,3,4,6,7,8,11,15,21,23,25,35,36)]

srf_vimp.explainer <- survex::explain(srf_ch_vimp.mod, verbose = F)

surv_pred <- srf_vimp.explainer$predict_survival_function(model = srf_vimp.explainer$model,
                                                          newdata = X_test_ch_vimp,
                                                          times = score_times)
```

```
risk <- srf_vimp.explainer$predict_function(model = srf_vimp.explainer$model,
                                             newdata = srf_vimp.explainer$data)

srf_ch_vimp.mod$C_idx <- c_index(y_true = srf_vimp.explainer$y, risk = risk)
srf_ch_vimp.mod$IBS = integrated_brier_score(y_test, surv = surv_pred, times = score_times)
srf_ch_vimp.mod$ICD_AUC =  integrated_cd_auc(y_test, surv = surv_pred, times = score_times)
srf_ch_vimp.mod$AIC <- NA


srf_ch_vimp.mod$C_idx
```

```
## [1] 0.8075157
```

```
srf_ch_vimp.mod$IBS
```

```
## [1] 0.09878886
```

```
srf_ch_vimp.mod$ICD_AUC
```

```
## [1] 0.7027414
```

```
srf_ch_vimp.mod$AIC
```

```
## [1] NA
```