An Investigation of Post-release Mortality of Coho Salmon in a Marine Recreational Fishery

Project Final Report

For

Emma Lunzmann-Cooke
Doctoral student, Forest and Conservation Sciences

by

Richard Hou[1,a)], Andy Lin[2,b)], Xiaotong Liu[2,c)], Weihao Qiu[2,d)]

[1]Department of Computer Science
University of British Columbia, Vancouver, British Columbia, Canada

[2]Department of Statistics
University of British Columbia, Vancouver, British Columbia, Canada

[a)]Corresponding author: ruijieh@student.ubc.ca
[b)]andy.lin@ubc.ca
[c)]xtlau@student.ubc.ca
[d)]qiu0805@student.ubc.ca

April 10, 2022

Summary

Fishery managers in catch-and-release fishing face a challenge in developing effective management tools to protect wild salmon populations. The objective of this statistical project is to examine the factors that influence fish survival after a catch-and-release event and make suggestions to minimize the mortality rate following the release. Researchers recorded 320 caught salmon, tagged and released them, to see whether they were detected (as survival). We have applied several classification models with different variable selection techniques to the cleaned and wrangled data, and found that logistic regression performs best among the classification models we analyzed, although not good enough with a classification accuracy of 0.70. In the best-performed logistic regression model, eye injury, hook location, fin damage, and reflex score are the key factors in predicting fish survival. The model selection relies on an integration of different measurements. Besides, our findings suggest that condensing several measurements into scores, such as injury score and reflex score, is inappropriate in this study.

Introduction

In recreational fisheries, a significant proportion of fish are released following capture, known as catch-and-release fishing. Released fish are frequently expected to survive; however, the actual catch-and-release mortality rates remain unknown for many marine fisheries, such as Pacific salmon (Pollock & PINE III, 2007). This research aims to identify the factors that influence coho salmon survival after a catch-and-release event in the ocean, as well as how these factors influence fish survival, so management tools and best fishing practices can be developed to reduce the mortality of wild fish.

Our statistical study addresses the following statistical questions: if a logistic regression is appropriate to determine the significant factors that influence fish survival after release; if model selection approaches, such as Akaike Information Criterion (AIC), are suitable in this study; whether condensing several measures into scores is more appropriate than involving them in the model independently. To address these questions, we generate many logistic regressions, compare their performance using various model selection criteria, and explore if there are any superior models and model selection approaches. In addition, we fit and compare different models using condensed scores and separate variables, respectively.

This report describes the data and statistical methods used, followed by the results of the analysis, including discussions of the preliminary findings, limitations, and future steps of the analysis.

Data Description

A tagging and tracking approach was employed to investigate the post-release mortality of coho salmon. In the summer of 2020 and 2021, 320 coho salmon were randomly caught and tagged with acoustic tags at various popular recreational fishing locations.The acoustic tags generated signals that were traced by acoustic receivers situated across the Salish Sea when the fish were swimming.

Predictor factors such as fish length, sex, air exposure time, injury score, reflex score, bleeding, and fat content were collected before the release of the salmon. Some variables are easy to understand from their names, while others are not. In particular, air exposure time measures how long the fish is held out of the water before tagging in seconds. The wound score is a numerical value indicating the severity of the injury of the fish. Scale loss indicates the extent to which the scale of the fish is lost in percentage.

To reduce the number of predictors, several metrics are condensed into scores. The injury score is the sum of 5 injury variables: fin damage, wound score, scale loss, eye injury, and bleeding. Similarly, the reflex score is a percentage that indicates the proportion of impaired reflexes among five reflexes: eye movement (VOR), orientation, body flex, tail flex, and venting. Fish survival is a binary response variable determined by whether the fish was detected or not detected after it was released using a set of acoustic receivers.

Among the 320 tagged and released salmon, only 281 that were guaranteed to go back to the receivers are reported in the data; those that were not heading to the region where they were born would not be detected for sure. We further cleaned the data by dropping the records with missing values. We found a relatively

small number of missing data, and the cause of missing data was random, so we discarded the salmon records with missing values and obtained 241 samples eventually.

Among all the 241 salmon, 145 are detected, indicating they survived, and 96 are not detected, so the response variable classes are relatively balanced. We have ten categorical predictors and four continuous predictors. To better comprehend the data and extract as many insights as possible, we have used a variety of graphs to visualize the relationship between each predictor and its response variable. **Figure 1** and **Figure 2** plot the frequency of each response variable level (detected and not detected) occurring at each level of the year and eye injury (categorical predictors), respectively. **Figure 3** and **4** contain parallel box plots for 2 continuous predictors (length and mean fat) as examples.

Figure 1. Frequency of each response variable level (detected and not detected) for different years (2020 and 2021). The percentage indicates the fraction of a certain level in a class. For example, the 0.61 in the plot above means that 61% of the salmon caught in 2020 were detected after release.

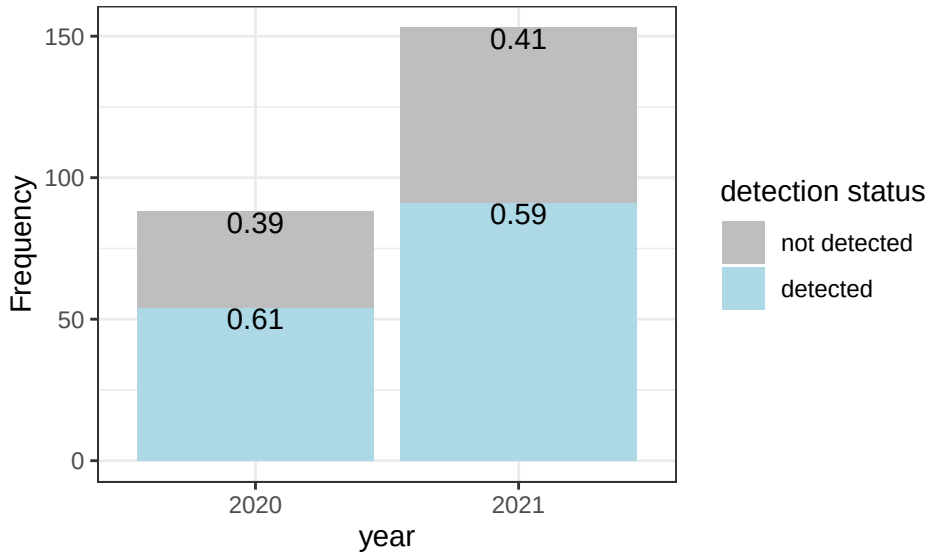


Figure 2. Frequency of each response variable level (detected and not detected) for each level of eye injury

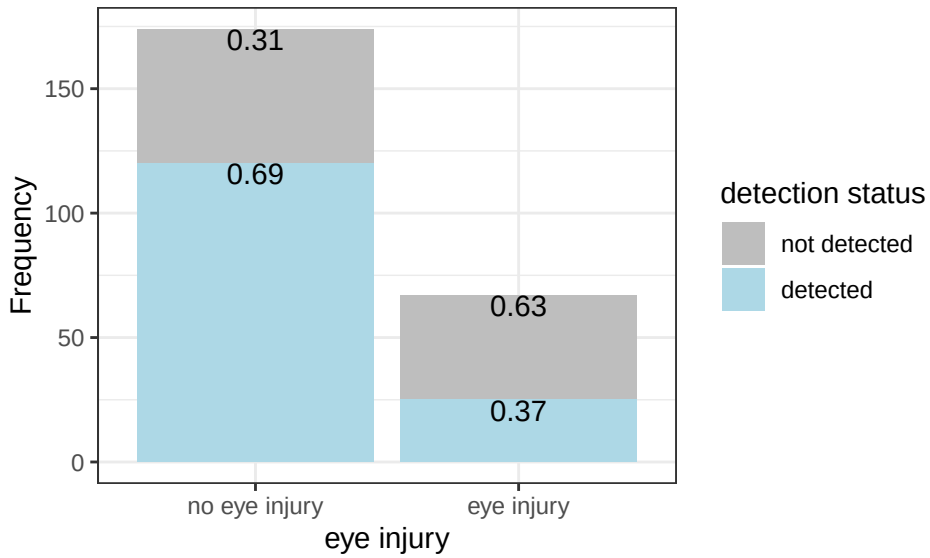


Figure 3. Parallel box plots for detection status (detected or not) versus fish length

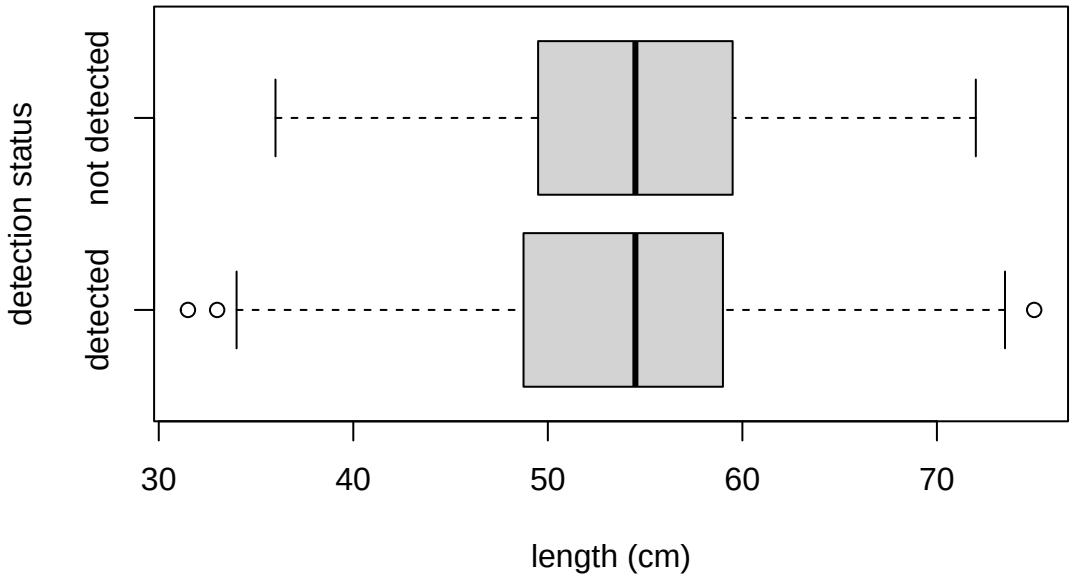
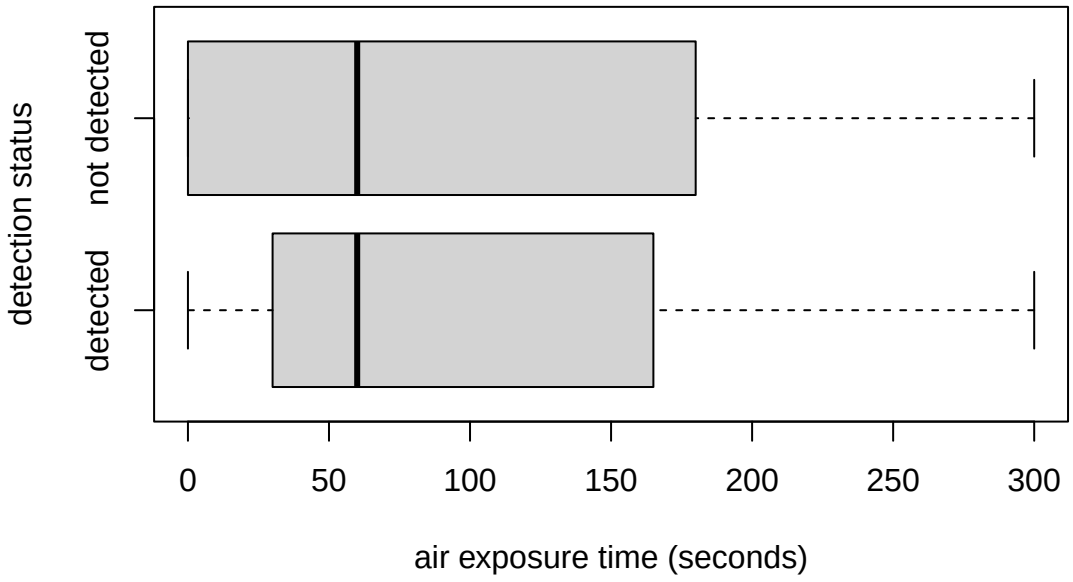


Figure 4. Parallel box plots for detection status (detected or not) versus air exposure time

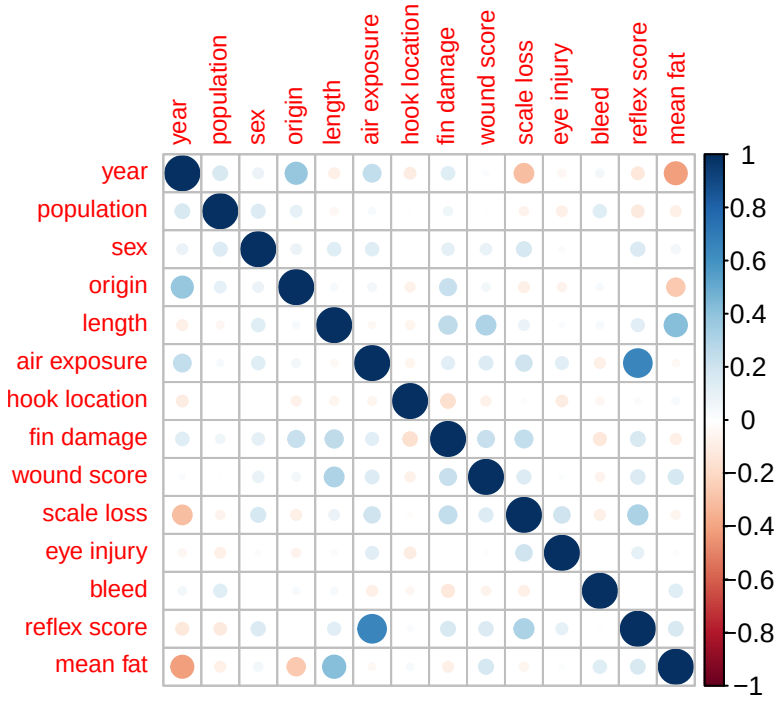


From **Figure 1**, we can spot that for some predictors (like the year), the fractions of salmon detected or not detected in each predictor level ('2020' and '2021' for the year) are close. We may presume that these factors do not have a significant role in determining fish survival. For certain other predictors (such as eye injury in **Figure 2**), the fractions of salmon detected or not detected are slightly or quite different in predictor levels ('0' and '1' for eye injury). We may anticipate these factors to be significant in the models. Similarly, for continuous predictors, we can see from **Figure 3** that the box plots for 'detected' and 'not detected' are similar, suggesting that the predictor variables (like fish length) might not be significant. In contrast, the different distributions of box plots in **Figure 4** might indicate that the predictors (like air exposure time) may be substantial.

Besides, to employ logistic regressions and other classification models later, we need to check the collinearity among the predictor variables. Thus, **Figure 5** below shows the correlations among these predictor variables,

to get a perspective of the relations among the predictors.

Figure 5. Correlation matrix of the predictor variables



We observe from **Figure 5** that the correlations, or multicollinearity, are relatively small (below 0.2) except the correlation between 'reflex score' and 'air exposure time' (around 0.66). We might remove one of the reflex score and air exposure time in the models if they are both identified as significant factors.

Methods

To answer the first statistical question about model and model selection, we built several logistic regressions and other classification models and compared their performance with different variable selection methods. To evaluate the performance of the fitted models on the new data, we used a large portion of the original data as training data to fit the models, and the remaining data as testing data to estimate the performance of the models.

Data Splitting

We divided the cleaned dataset into training and testing sets in a 3:1 ratio. To maintain the proportions of each level of the response variable in both training and testing data, the data were split stratified by detection status (detected or not detected).

Logistic Regression and AIC

Following data splitting, we built classification models to predict whether the fish would survive a catch-and-release event based on the values of the predictors. Previous research suggested that logistic regression was the most effective for similar challenges based on the binary categorical response variable and continuous and categorical predictors (Nelson, 1998), so we evaluated several logistic regressions with variable selection. All the models used Akaike Information Criterion (AIC) to measure and compare the models' performance. Lower AIC values indicate a better-fit model, which has a higher classification accuracy on training data.

To learn about the feasibility of the models, we built the full logistic regression using all the predictors and stepwise logistic regressions to reduce the number of predictors and select the significant ones. Stepwise

regressions select independent variables to use in a model with the lowest AIC based on an iterative process of adding or removing predictors. Since stepwise regressions did not cover all subsets of the predictors, we applied significance testing to the model and built a new model using the statistically significant predictor variables.

Alternative Models and Model Selections (Cross-Validation)

In addition to logistic regressions and AIC, we explored alternative classification models and model selection approaches. Since logistic regressions only support solving linear relations between predictors and the response variables, we have tried other models supporting non-linear solutions, such as k-nearest neighbors and random forests, to test whether there are more complicated relations among the variables.

We evaluated other model selection methods like BIC and 5-fold cross-validation as well. To develop a 5-fold cross-validation, we split the original training data into 5 groups. We utilized 4 groups as new training data to fit the models and the other group as validation data to evaluate the models' performance. Since the logistic regression prediction function returns a probability between 0 and 1, we need to define a threshold, called decision boundary, to determine which class the data belong to. The record is categorized as "detected" if the estimated probability is above the decision boundary; otherwise, it is labelled as "not detected". After fitting the models with training data, we tweaked the decision boundaries to identify the classifier with the best training accuracy, which indicates the classification accuracy in the training data. Then, we employed the recognized model to validation data and compute the validation accuracy, the classification accuracy of the response variable (to be detected or not) in validation data. We repeated this 5 times, each with different training and validation data splitting. We then selected the model with the highest mean validation accuracy.

Condensed Scores or Separate Variables

To address the second statistical question about whether to condense variables into scores where possible, we applied the model construction and selection methods mentioned above to both condensed scores and separate variables. Based on the original dataset, we first summed five injury predictors' values (fin damage, wound score, scale loss, eye injury, and bleeding) into one injury score. Then we built the same models and compared the models' performance. Secondly, based on the original dataset, we expanded the reflex score, which was condensed from five reflex predictors (eye movement, orientation, body flex, tail flex, and venting). We included these predictors separately in the models and compared the models' performance.

Results

The results are reported in two sections: selecting classification models and model selection methods; evaluating the models using separate variables and condensed scores, respectively. These parts are based on themes that emerged as major discussion areas during the analysis process.

Models and Model Selection

For the first question about models and model selection approaches, **Table 1** below is a summary of some representative models we have fitted.

Table 1. Summary of Some Representative Models Trained

Models	AIC	Training Accuracy	Validation Accuracy	Testing Accuracy
Full Logistic	227.49	0.788	0.639	0.606
Stepwise Logistic	213.02	0.771	0.650	0.623
Logistic with Significant Test	220.64	0.744	0.655	0.639
Logistic removing imbalance	218.45	0.732	0.672	0.656
Best Logistic	217.99	0.721	0.707	0.672

As **Table 1** shows, the AIC of the full model is 227.49. Training accuracy and validation accuracy are the classification accuracy of the response variable in training and validation data, respectively. The highest

training accuracy of the full model is 0.788, and the corresponding validation accuracy is 0.639. The forward and backward stepwise logistic regressions give us the same model and the same AIC, 213.02, which is better than the full model. The highest training accuracy of this model is 0.771, and the corresponding validation accuracy is 0.650. The validation accuracy agrees with the AIC conclusion that the reduced model is superior to the full model.

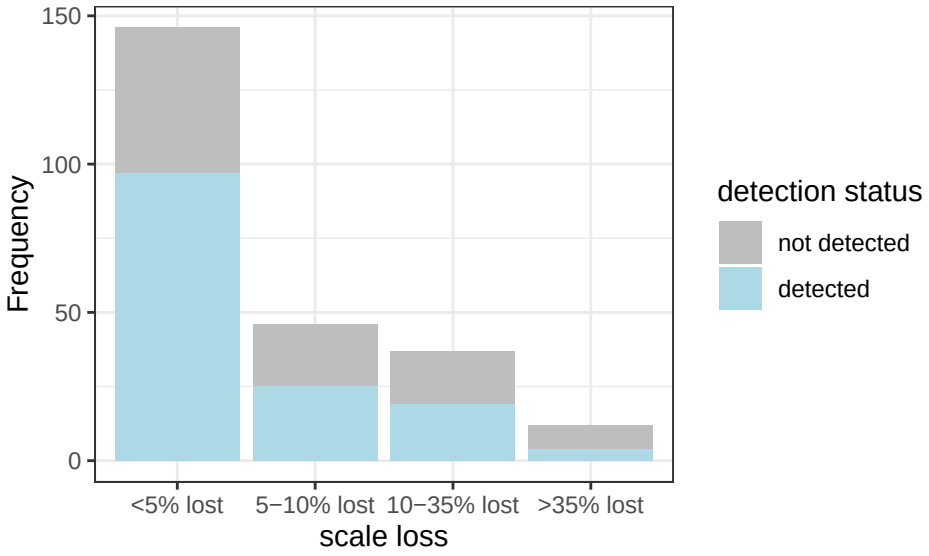
The stepwise regressions may have missed some combinations of the predictor variables. We can see from the full model that some of the predictors selected in the stepwise regressions, like 'year' and 'origin', are not statistically significant, so the significance testing is applied. The testing indicates that eye injury, hook location, fin damage, air exposure time, scale loss and reflex score are significant, as shown in **Table 2** below. The model built with these factors has an AIC of 220.64, higher than the stepwise regression models. The highest training accuracy of this model is 0.744, and the validation accuracy is 0.655. From the perspective of AIC, the stepwise regression model is better with lower AIC. However, the significant model has lower training accuracy and higher validation accuracy. This finding may suggest that the stepwise regression model overfits and using the model selection criterion like AIC alone might not be appropriate.

Table 2. Interpretations (Selected Variables) of Representative Models Stated before

Models	**Variables Selected**
Logistic with Significant Test	eye injury, hook location, fin damage, air exposure time, scale loss, reflex score
Logistic removing imbalance	eye injury, hook location, fin damage, air exposure time, reflex score
Best Logistic	eye injury, hook location, fin damage, reflex score

We have found that the scale loss is significant mostly because level 3 of scale loss is significant. However, as **Figure 6** below shows, we can see that the number of salmon in level 3 is relatively negligible. Considering this as significant with the limited number of data might not be appropriate, so we have removed scale loss from the significance test and built a new model with an AIC of 218.45. Similarly, we can see that this model performs better than the significant model and prevents overfitting based on validation accuracy, as can be seen in **Table 1**. It can be interpreted that the fish, which has a higher probability to survive, would have no eye injury, get hooked at the bottom jaw instead of the top jaw, have a minor fraying fin, be exposed in the air longer, and have a lower reflex score.

Figure 6. Frequency of each response variable level (detected and not detected) for different scale loss extent of the fish



Further, as we interpret above, if the fish is exposed to the air longer, it is more likely to survive after release, which contradicts our intuition. We guess that the issue comes from the data itself. According to **Figure 4**, the boxplots for detected and not detected are relatively similar; then, the air exposure time can be slightly positively significant. We see that the air exposure time has a minor effect (smallest weight in the model). Another reason might be that the correlation between reflex score and air exposure time is high. Based on the two findings above, we suggest dropping air exposure. Thus, as shown in **Table 2**, we select the logistic regression including eye injury, hook location, fin damage, and reflex score as our current best model. It can be interpreted that the fish, which has a higher probability to survive, would have no eye injury, get hooked at the bottom jaw instead of the top jaw, have a minor fraying fin, and have a lower reflex score.

As **Table 1** shows, when we evaluate the models on testing data, the results are consistent with our model selection. The best logistic regression we selected has the highest classification accuracy of around 0.67 on testing data, which suggests that our selected model performs well in predicting new records.

In addition, we have fitted several classification models and alternative model selection methods. Among all the models we have tried, except for logistic regressions, the model that performs best is Random Forest, with a validation accuracy of 0.656. It is found that other model selection approaches such as BIC and adjusted R^2 are not suitable for our study.

Condensed Scores or Separate Variables

As for the second statistical question, we want to determine if condensing several measurements into scores is the optimal strategy, or if the variables should all be independently involved in the model. We have applied all the analysis processes to the new data to accomplish this task. First, we have condensed five categorical predictors into one new injury score by summation. Under the best model, it has lower classification accuracy on validation data (0.653). Secondly, we have expanded the reflex score and fitted models with the separate five reflex variables. Under the best model, it has slightly higher classification accuracy on validation data (0.708) than before.

The variables selected are eye injury, hook location, fin damage, air exposure, and orientation reflex. Recall that the orientation reflex is one of the reflex variables condensed into the reflex score. We might choose this model as our best model instead of the one with the condensed reflex score. It can be interpreted that the fish, which has a higher probability to survive, would have no eye injury, get hooked at the bottom jaw instead of the top jaw, have a minor fraying fin, be exposed in the air longer, and have lower orientation reflex injury.

Conclusions

An analysis of the data and findings suggests that eye injury, hook location, fin damage, and reflex score (or orientation reflex in particular) are likely to be the key factors influencing coho salmon survival following a catch-and-release event in the ocean. Based on these factors, we suggest the fishery managers control the time of fish exposed to the air, use safer instruments while fishing to protect the fish fin and scales, and release salmon with no eye injury, no orientation reflex impaired, bottom jaw hooked, and minor fraying fin.

The logistic regression is appropriate to determine the statistical significance of the factors since the significant model, built from the significant predictors, performs better than the full model and stepwise regression models. In terms of the accuracy of around 0.70, however, the logistic regression is not good enough in predicting fish survival in practice. A large number of salmon would die after release, even though they are expected to survive. Further, other classification algorithms like K-Nearest Neighbors (KNNs) and Random Forest did not outperform our best logistic regression. Besides, using model selection approaches such as Akaike Information Criterion (AIC) alone is unsuitable and probably even dangerous since the lower AIC may lead to overfitting. We suggest combining several measurements, such as using AIC and k-fold cross-validation together to select the statistically significant factors, as well as prevent overfitting.

To reduce the number of predictor variables from the models, sometimes it can be tempting to condense several measurements into scores where possible. According to the results, we have concluded that it might not be helpful in this study, at least for injury score and reflex score. First, if the variables to be merged are not equally important, condensing might cause factors to be less significant than they should be. Secondly,

it would be hard or unreasonable to interpret the selected predictors. For example, if we replace the eye injury and fin damage with injury score, we would not know what kinds of injury count, leading to wrong suggestions for fishery managers.

Further Discussions

In the context of this report, it must also be mentioned that there exist certain limitations in the data collection and statistical methods. The classes are highly imbalanced for some categorical predictors such as wound score and hook location. As is shown in **Figure 6**, some levels have more than one hundred records (level 0 for scale loss), while some have less than five records (level 3 for scale loss). The classes with a pretty limited number of salmon would affect the weight of other classes in predicting and lead to wrong predictions of the salmon in these classes. We suggest removing the imbalance by either dropping the classes with only a few salmon records fill in or combining categories in some reasonable ways.

Moreover, the original data are randomly divided into training and testing data. We obtain slightly different models' performances when splitting the data in different ways and training the models following the same process. It would be better to run more experiments with different training and testing data splitting. The sample size is limited as well, which might be an essential factor leading to the limited classification accuracy. With more data collected, we believe that the logistic regression we have selected will perform much better following the same analysis process.

References

Nelson, K. L. (1998). Catch-and-release mortality of striped bass in the Roanoke River, North Carolina. *North American Journal of Fisheries Management, 18*(1), 25-30.

Pollock, K. H., & PINE III, W. E. (2007). The design and analysis of field studies to estimate catch-and-release mortality. *Fisheries Management and Ecology, 14*(2), 123-130.

Appendix

Read data

```
# read data and perspective
raw_data <- read_xlsx("Lunzmann-Cooke_data_20210124.xlsx", sheet = "data", col_names = TRUE)
head(raw_data); dim(raw_data)
```

Some columns incorrectly converted to char() due to NA

```
# correct the variables data type
# Warning: note that NAs introduced by coercion
raw_data <- raw_data %>%
                mutate(air.exposure = as.double(air.exposure)) %>%
                mutate(hook.location = as.double(hook.location)) %>%
                mutate(mean.fat = as.double(mean.fat))
```

Removing all the NA's and Unknown's as missing data (40 in total)

```
# remove the missing data
complete_data <- raw_data %>%
                  filter(!is.na(air.exposure) & !is.na(hook.location) & !is.na(mean.fat) &
                         (tolower(population) != "unknown") & (tolower(sex) != "unknown") )
```

Refine the data table (remove fish.no which is equivalent to tag.id)

```
# remove fish.no column and check that tag.id is an unique identifier
complete_data <- complete_data %>%
                    select(-"fish.no")
all(complete_data$tag.id == unique(complete_data$tag.id))
```

Change the date to year

```
complete_data <- complete_data %>%
                    mutate(year=substr(date,1,4), .after=date)
```

Preliminary EDA

Number of each detection.status

```
complete_data %>% group_by(detection.status) %>% summarise(Count = n())
# 96 of 241 not detected (mortality), 145 of 241 detected (survival)
```

14 predictors (10 categorical and 4 continuous) and 1 response variable

Make Frequency Plots for Categorical predictors:

year, population, sex, origin, hook.location, fin.damage, wound.score, scale.loss, eye.injury, bleed

```
# define the functions used in visualization
fraction <- function(Freq){
  range = c(1:length(Freq))
  for (i in range){
    if ((i %% 2) != 0){
```

```
      a = Freq[i]/(Freq[i]+Freq[i+1])
      b = Freq[i+1]/(Freq[i]+Freq[i+1])
      Freq[i]=a
      Freq[i+1]=b
    }
  }
  return(round(Freq,2))
}
bar_plot <- function(table, name_table){
  colnames(name_table) <- c("not detected","detected")
  row.names(table) <- c("not detected","detected")
  namelist<-unlist(strsplit(names(dimnames(name_table))[1], "\\."))
  if (length(namelist) ==1){
    x_labs <- names(dimnames(name_table))[1]
  } else {
    x_labs <- paste(namelist[1],namelist[2], sep=" ")
  }
  ggplot(as.data.frame(table),aes(x=Var2,y=Freq,fill=Var1)) +
      geom_bar(stat="identity",position="stack")+
      xlab(x_labs) + ylab("Frequency") + labs(fill="detection status")+
      geom_text(aes(label=fraction(Freq)),position="stack",vjust=1)+
      scale_fill_manual(values=c("grey","light blue"))+
      theme_bw()
}
```

For year:

```
# For year:
# data summary: 88 in year 2020, 153 in year 153
vs.year <- table(complete_data$year, complete_data$detection.status)
names(dimnames(vs.year)) <- c("year", "detection.status")
addmargins(vs.year)
# data visualization:
year_table <- table(complete_data$detection.status, complete_data$year)
bar_plot(year_table, vs.year)
```

For population:

```
# For population:
vs.popu <- table(complete_data$population, complete_data$detection.status)
names(dimnames(vs.popu)) <- c("population", "detection.status")
addmargins(vs.popu)
# data visualization:
popu_table <- table(complete_data$detection.status, complete_data$population)
bar_plot(popu_table, vs.popu)
```

For sex:

```
# For sex:
vs.sex <- table(complete_data$sex, complete_data$detection.status)
names(dimnames(vs.sex)) <- c("sex", "detection.status")
addmargins(vs.sex)
```

```r
# data visualization:
sex_table <- table(complete_data$detection.status, complete_data$sex)
bar_plot(sex_table, vs.sex)
```

For origin:

```r
# For origin:
vs.origin <- table(complete_data$origin, complete_data$detection.status)
names(dimnames(vs.origin)) <- c("origin", "detection.status")
addmargins(vs.origin)
# data visualization:
origin_table <- table(complete_data$detection.status, complete_data$origin)
bar_plot(origin_table, vs.origin)
```

For hook.location:

```r
# For hook.location:
vs.hook <- table(complete_data$hook.location, complete_data$detection.status)
names(dimnames(vs.hook)) <- c("hook.location", "detection.status")
addmargins(vs.hook)
# data visualization:
hook_table <- table(complete_data$detection.status, complete_data$hook.location)
bar_plot(hook_table, vs.hook)
```

For fin.damage:

```r
# For fin.damage:
vs.fin_damage <- table(complete_data$fin.damage, complete_data$detection.status)
names(dimnames(vs.fin_damage)) <- c("fin.damage", "detection.status")
addmargins(vs.fin_damage)
# data visualization:
fin_table <- table(complete_data$detection.status, complete_data$fin.damage)
bar_plot(fin_table, vs.fin_damage)
```

For wound.score:

```r
# For wound.score:
vs.wound <- table(complete_data$wound.score, complete_data$detection.status)
names(dimnames(vs.wound)) <- c("wound.score", "detection.status")
addmargins(vs.wound)
# data visualization:
wound_table <- table(complete_data$detection.status, complete_data$wound.score)
bar_plot(wound_table, vs.wound)
```

For scale.loss:

```r
# For scale.loss:
vs.scale <- table(complete_data$scale.loss, complete_data$detection.status)
names(dimnames(vs.scale)) <- c("scale.loss", "detection.status")
addmargins(vs.scale)
# data visualization:
```

```
scale_table <- table(complete_data$detection.status, complete_data$scale.loss)
bar_plot(scale_table, vs.scale)
```

For eye.injury:

```
# For eye.injury:
vs.eye <- table(complete_data$eye.injury, complete_data$detection.status)
names(dimnames(vs.eye)) <- c("eye.injury", "detection.status")
addmargins(vs.eye)
# data visualization:
eye.injury_table <- table(complete_data$detection.status, complete_data$eye.injury)
bar_plot(eye.injury_table, vs.eye)
```

For bleed:

```
# For bleed:
vs.bleed <- table(complete_data$bleed, complete_data$detection.status)
names(dimnames(vs.bleed)) <- c("bleed", "detection.status")
addmargins(vs.bleed)
# data visualization:
bleed_table <- table(complete_data$detection.status, complete_data$bleed)
bar_plot(bleed_table, vs.bleed)
```

Make Boxplots for Continuous predictors:

length, air.exposure, reflex.score, mean.fat

For length:

```
# For length:
# data summary:
summary(complete_data$length)

# data visualization:
plot(complete_data$length, complete_data$detection.status)
boxplot(length~detection.status, data=complete_data, horizontal=TRUE)
```

For air.exposure:

```
# For air.exposure:
# data summary:
summary(complete_data$air.exposure)

# data visualization:
plot(complete_data$air.exposure, complete_data$detection.status)
boxplot(air.exposure~detection.status, data=complete_data, horizontal=TRUE)
```

For reflex.score:

```
# For reflex.score:
# data summary:
summary(complete_data$reflex.score)
```

```
# data visualization:
plot(complete_data$reflex.score, complete_data$detection.status)
boxplot(reflex.score~detection.status, data=complete_data, horizontal=TRUE)
```

For mean.fat:

```
# For mean.fat:
# data summary:
summary(complete_data$mean.fat)

# data visualization:
plot(complete_data$mean.fat, complete_data$detection.status)
boxplot(mean.fat~detection.status, data=complete_data, horizontal=TRUE)
```

Correlation matrix

```
## For correlation and plot correlation matrix
complete_data_1 <- complete_data %>%
                    mutate(year = ifelse(year == 2020,0,1)) %>%
                    mutate(sex = ifelse(sex == "male",0,1)) %>%
                    mutate(origin = ifelse(origin == "w", 0, 1))

complete_data_1 <- complete_data_1 %>%
                    mutate(population = case_when(population == "BB" ~ 1, population == "ECVI" ~2,
                                                  population == "FR" ~ 3,
                                        population == "HB" ~ 4, population == "PS" ~ 5))

correlation <- cor(complete_data_1[,3:17]) # involve the response variable
#correlation <- cor(complete_data_1[,3:16])

corrplot::corrplot(correlation, tl.cex = 0.75)
```

Model fitting

Factor the categorical variables

```
## Factor
fit_dataset<-complete_data %>%
                  mutate(year=as.factor(year)) %>%
                  mutate(population=as.factor(population)) %>%
                  mutate(sex=as.factor(sex)) %>%
                  mutate(origin=as.factor(origin)) %>%
                  mutate(hook.location=as.factor(hook.location)) %>%
                  mutate(fin.damage=as.factor(fin.damage)) %>%
                  mutate(wound.score=as.factor(wound.score)) %>%
                  mutate(scale.loss=as.factor(scale.loss)) %>%
                  mutate(eye.injury=as.factor(eye.injury)) %>%
                  mutate(bleed=as.factor(bleed)) %>%
                  mutate(detection.status=as.factor(detection.status)) %>%
                  #mutate(air.exposure=as.factor(air.exposure)) %>%
                  select(-"tag.id") %>%
                  select(-"date")
```

```
str(fit_dataset)
```

Split into training dataset and testing dataset (4:1) proportional to detection.status

```
# Sample from detected:
detected <- fit_dataset %>% filter(detection.status == 1)
smp_size <- floor(0.75 * nrow(detected))
set.seed(450)
detected_train <- sample(seq_len(nrow(detected)), size = smp_size)

train_det <- detected[detected_train, ]
test_det <- detected[-detected_train, ]

# Sample from notdetected:
notdetected <- fit_dataset %>% filter(detection.status == 0)
smp_size <- floor(0.75 * nrow(notdetected))
set.seed(451)
notdetected_train <- sample(seq_len(nrow(notdetected)), size = smp_size)

train_notdet <- notdetected[notdetected_train, ]
test_notdet <- notdetected[-notdetected_train, ]

# Combine detected and notdetected
train <- rbind(train_det, train_notdet)
test <- rbind(test_det, test_notdet)

# Shuffle the rows
#set.seed(452)
rows_train <- sample(nrow(train))
train <- train[rows_train, ]

#set.seed(453)
rows_test <- sample(nrow(test))
test <- test[rows_test, ]
```

Fit logistic regression using training data (full_model, forwards, backwards, and their AIC)

Full model

```
# The full model
fullmodel <- glm(detection.status~.,family=binomial(link='logit'),data=train)
summary(fullmodel)
```

AIC = 227.49

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.fullmodel <- predict(fullmodel,newdata=subset(train,select=seq(1,14)),type='response')
  train.fullmodel <- ifelse(train.fullmodel > 0.1*k,1,0)
  misClasificError <- mean(train.fullmodel != train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Full model with k-fold cross validation

```
# split the training data into five parts
partition <- 180/5 # =36
folds <- c(1:36, 37:72, 73:108, 109:144, 145:180)

# Cross-Validation
range.fold <- c(1,2,3,4,5)
fold_acc <- c(1,2,3,4,5)
train_acc <- c(1,2,3,4,5)
for (k in range.fold){
  train.data <- train[-((36*(k-1)+1):(36*k)),]
  train.model <- glm(detection.status~.,
                     family=binomial(link='logit'),data=train.data)

  # calculate validation accuracy:

  # check the categorical variables levels in validation data
  # remove the 'new' levels in validation data
  valid.variables <- train[(36*(k-1)+1):(36*k),1:14]
  valid.data <- train[(36*(k-1)+1):(36*k),]
  for (var in variable.names(train.data)){
    train.var <- pull(train.data[var])
    if (is.factor(train.var)){
      train.levels <- unique(train.var)
      valid.var <- pull(valid.data[var])
      valid.levels <- unique(valid.var)
      if (!(all(valid.levels %in% train.levels))){
        # find the new level(s) in valid but not in train
        new.levels <- valid.levels[!(valid.levels %in% train.levels)]
        valid.variables <- valid.variables[which(valid.var %in% train.levels),]
        valid.data <- valid.data[which(valid.var %in% train.levels),]
      }
    }
  }

  range.decision_boundary <- c(0,1,2,3,4,5,6,7,8,9,10)
  accs <- c(0,1,2,3,4,5,6,7,8,9,10)
  for (i in range.decision_boundary){
    # training accuracy for different decision boundaries:
    valid.model <- predict(train.model,newdata=train.data, type='response')
    valid.model <- ifelse(valid.model > 0.1*i,1,0)
    misClasificError <- mean(valid.model != train.data$detection.status)
    accs[i+1] <- 1-misClasificError
  }
  # selection the max accuracy
  train_acc[k] <- max(accs)
  best_bound <- which.max(accs)

  accss <- c(1,2,3)
  # compute the validation accuracy
  valid.model <- predict(train.model,newdata=valid.variables, type='response')
  valid.model <- ifelse(valid.model > 0.1*(best_bound-1),1,0)
  misClasificError <- mean(valid.model != valid.data$detection.status)
```

```
  accss[1] <- 1-misClasificError

  valid.model <- predict(train.model,newdata=valid.variables, type='response')
  valid.model <- ifelse(valid.model > 0.1*best_bound,1,0)
  misClasificError <- mean(valid.model != valid.data$detection.status)
  accss[2] <- 1-misClasificError

  valid.model <- predict(train.model,newdata=valid.variables, type='response')
  valid.model <- ifelse(valid.model > 0.1*(best_bound+1),1,0)
  misClasificError <- mean(valid.model != valid.data$detection.status)
  accss[3] <- 1-misClasificError

  fold_acc[k] <- max(accss)
}
# calculate the mean of the max accuracies for the folds
acccuracy_train <- mean(train_acc)
acccuracy_cv <- mean(fold_acc)
# outputs
train_acc; acccuracy_train
fold_acc; acccuracy_cv
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.fullmodel <- predict(fullmodel,newdata=subset(test,select=seq(1,14)),type='response')
  test.fullmodel <- ifelse(test.fullmodel > 0.1*k,1,0)
  misClasificError <- mean(test.fullmodel != test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Backward Selection

```
# Backward Selection
backwards = stats::step(fullmodel)
summary(backwards)
formula(backwards)
```

AIC = 213.02

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.backmodel <- predict(backwards,newdata=subset(train,select=seq(1,14)),type='response')
  train.backmodel <- ifelse(train.backmodel > 0.1*k,1,0)
  misClasificError <- mean(train.backmodel != train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.backmodel <- predict(backwards,newdata=subset(test,select=seq(1,14)),type='response')
```

```
  test.backmodel <- ifelse(test.backmodel > 0.1*k,1,0)
  misClasificError <- mean(test.backmodel != test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Forward Selection

```
# Forward Selection
basemodel <- glm(detection.status~NULL, family=binomial(link='logit'),data=train);
forwards <- stats::step(basemodel,scope=list(lower=formula(basemodel),upper=formula(fullmodel)), direct
summary(forwards)
formula(forwards)
```

AIC = 213.02

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.foremodel <- predict(forwards,newdata=subset(train,select=seq(1,14)),type='response')
  train.foremodel <- ifelse(train.foremodel > 0.1*k,1,0)
  misClasificError <- mean(train.foremodel != train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Stepwise regression model with k-fold cross validation

```
# split the training data into five parts
partition <- 180/5 # =36
folds <- c(1:36, 37:72, 73:108, 109:144, 145:180)

# Cross-Validation
range.fold <- c(1,2,3,4,5)
fold_acc <- c(1,2,3,4,5)
train_acc <- c(1,2,3,4,5)
for (k in range.fold){
  train.data <- train[-((36*(k-1)+1):(36*k)),]
  train.model <- glm(detection.status~year + origin + air.exposure + hook.location +
                       fin.damage + scale.loss + eye.injury + reflex.score,
                     family=binomial(link='logit'),data=train.data)

  # calculate validation accuracy:

  # check the categorical variables levels in validation data
  # remove the 'new' levels in validation data
  valid.variables <- train[(36*(k-1)+1):(36*k),1:14]
  valid.data <- train[(36*(k-1)+1):(36*k),]
  for (var in variable.names(train.data)){
    train.var <- pull(train.data[var])
    if (is.factor(train.var)){
      train.levels <- unique(train.var)
      valid.var <- pull(valid.data[var])
      valid.levels <- unique(valid.var)
      if (!(all(valid.levels %in% train.levels))){
```

```
        # find the new level(s) in valid but not in train
        new.levels <- valid.levels[!(valid.levels %in% train.levels)]
        valid.variables <- valid.variables[which(valid.var %in% train.levels),]
        valid.data <- valid.data[which(valid.var %in% train.levels),]
      }
    }
  }

  range.decision_boundary <- c(0,1,2,3,4,5,6,7,8,9,10)
  accs <- c(0,1,2,3,4,5,6,7,8,9,10)
  for (i in range.decision_boundary){
    # training accuracy for different decision boundaries:
    valid.model <- predict(train.model,newdata=train.data, type='response')
    valid.model <- ifelse(valid.model > 0.1*i,1,0)
    misClasificError <- mean(valid.model != train.data$detection.status)
    accs[i+1] <- 1-misClasificError
  }
  # selection the max accuracy
  train_acc[k] <- max(accs)
  best_bound <- which.max(accs)

  # compute the validation accuracy
  valid.model <- predict(train.model,newdata=valid.variables, type='response')
  valid.model <- ifelse(valid.model > 0.1*best_bound,1,0)
  misClasificError <- mean(valid.model != valid.data$detection.status)
  fold_acc[k] <- 1-misClasificError
}
# calculate the mean of the max accuracies for the folds
acccuracy_train <- mean(train_acc)
acccuracy_cv <- mean(fold_acc)
# outputs
train_acc; acccuracy_train
fold_acc; acccuracy_cv
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.foremodel <- predict(forwards,newdata=subset(test,select=seq(1,14)),type='response')
  test.foremodel <- ifelse(test.foremodel > 0.1*k,1,0)
  misClasificError <- mean(test.foremodel != test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Statistical Significant Predictors

```
# Variable selection according to test
anova(fullmodel, test="Chisq")
sel_model <- glm(detection.status~eye.injury+hook.location+scale.loss+reflex.score+fin.damage+air.exposu
summary(sel_model)
formula(sel_model)
```

AIC = 220.64

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.sel_model <- predict(sel_model,newdata=subset(train,select=seq(1,14)),type='response')
  train.sel_model <- ifelse(train.sel_model > 0.1*k,1,0)
  misClasificError <- mean(train.sel_model != train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

significance model with k-fold cross validation

```
# split the training data into five parts
partition <- 180/5 # =36
folds <- c(1:36, 37:72, 73:108, 109:144, 145:180)

# Cross-Validation
range.fold <- c(1,2,3,4,5)
fold_acc <- c(1,2,3,4,5)
train_acc <- c(1,2,3,4,5)
for (k in range.fold){
  train.data <- train[-((36*(k-1)+1):(36*k)),]
  train.model <- glm(detection.status~air.exposure + hook.location + fin.damage
                     + scale.loss + eye.injury + reflex.score,
                      family=binomial(link='logit'),data=train.data)

  # calculate validation accuracy:

  # check the categorical variables levels in validation data
  # remove the 'new' levels in validation data
  valid.variables <- train[(36*(k-1)+1):(36*k),1:14]
  valid.data <- train[(36*(k-1)+1):(36*k),]
  for (var in variable.names(train.data)){
    train.var <- pull(train.data[var])
    if (is.factor(train.var)){
      train.levels <- unique(train.var)
      valid.var <- pull(valid.data[var])
      valid.levels <- unique(valid.var)
      if (!(all(valid.levels %in% train.levels))){
        # find the new level(s) in valid but not in train
        new.levels <- valid.levels[!(valid.levels %in% train.levels)]
        valid.variables <- valid.variables[which(valid.var %in% train.levels),]
        valid.data <- valid.data[which(valid.var %in% train.levels),]
      }
    }
  }

  range.decision_boundary <- c(0,1,2,3,4,5,6,7,8,9,10)
  accs <- c(0,1,2,3,4,5,6,7,8,9,10)
  for (i in range.decision_boundary){
    # training accuracy for different decision boundaries:
    valid.model <- predict(train.model,newdata=train.data, type='response')
    valid.model <- ifelse(valid.model > 0.1*i,1,0)
```

```
    misClasificError <- mean(valid.model != train.data$detection.status)
    accs[i+1] <- 1-misClasificError
  }
  # selection the max accuracy
  train_acc[k] <- max(accs)
  best_bound <- which.max(accs)

  # compute the validation accuracy
  accss <- c(1,2,3)
  for (i in accss){
    valid.model <- predict(train.model,newdata=valid.variables, type='response')
    valid.model <- ifelse(valid.model > 0.1*(best_bound-2+i),1,0)
    misClasificError <- mean(valid.model != valid.data$detection.status)
    accss[i] <- 1-misClasificError
  }
  fold_acc[k] <- max(accss)
}
# calculate the mean of the max accuracies for the folds
acccuracy_train <- mean(train_acc)
acccuracy_cv <- mean(fold_acc)
# outputs
train_acc; acccuracy_train
fold_acc; acccuracy_cv
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.sel_model <- predict(sel_model,newdata=subset(test,select=seq(1,14)),type='response')
  test.sel_model <- ifelse(test.sel_model > 0.1*k,1,0)
  misClasificError <- mean(test.sel_model != test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Statistical Significant Predictors removing scale loss

```
# Variable selection according to test
anova(fullmodel, test="Chisq")
sel_model <- glm(detection.status~eye.injury+hook.location+reflex.score+fin.damage+air.exposure, family=
summary(sel_model)
formula(sel_model)
```

AIC = 218.45

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.sel_model <- predict(sel_model,newdata=subset(train,select=seq(1,14)),type='response')
  train.sel_model <- ifelse(train.sel_model > 0.1*k,1,0)
  misClasificError <- mean(train.sel_model != train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Removing scale loss with k-fold cross validation

```
# split the training data into five parts
partition <- 180/5 # =36
folds <- c(1:36, 37:72, 73:108, 109:144, 145:180)

# Cross-Validation
range.fold <- c(1,2,3,4,5)
fold_acc <- c(1,2,3,4,5)
for (k in range.fold){
  train.data <- train[-((36*(k-1)+1):(36*k)),]
  train.model <- glm(detection.status~eye.injury+hook.location+fin.damage+reflex.score+air.exposure,
                     family=binomial(link='logit'),data=train.data)

  # calculate validation accuracy:

  # check the categorical variables levels in validation data
  # remove the 'new' levels in validation data
  valid.variables <- train[(36*(k-1)+1):(36*k),1:14]
  valid.data <- train[(36*(k-1)+1):(36*k),]
  for (var in variable.names(train.data)){
    train.var <- pull(train.data[var])
    if (is.factor(train.var)){
      train.levels <- unique(train.var)
      valid.var <- pull(valid.data[var])
      valid.levels <- unique(valid.var)
      if (!(all(valid.levels %in% train.levels))){
        # find the new level(s) in valid but not in train
        new.levels <- valid.levels[!(valid.levels %in% train.levels)]
        valid.variables <- valid.variables[which(valid.var %in% train.levels),]
        valid.data <- valid.data[which(valid.var %in% train.levels),]
      }
    }
  }

  range.decision_boundary <- c(0,1,2,3,4,5,6,7,8,9,10)
  accs <- c(0,1,2,3,4,5,6,7,8,9,10)
  for (i in range.decision_boundary){
    # training accuracy for different decision boundaries:
    valid.model <- predict(train.model,newdata=train.data, type='response')
    valid.model <- ifelse(valid.model > 0.1*i,1,0)
    misClasificError <- mean(valid.model != train.data$detection.status)
    accs[i+1] <- 1-misClasificError
  }
  # selection the max accuracy
  train_acc[k] <- max(accs)
  best_bound <- which.max(accs)

  # compute the validation accuracy
  accss <- c(1,2,3)
  for (i in accss){
    valid.model <- predict(train.model,newdata=valid.variables, type='response')
    valid.model <- ifelse(valid.model > 0.1*(best_bound-2+i),1,0)
    misClasificError <- mean(valid.model != valid.data$detection.status)
```

```
    accss[i] <- 1-misClasificError
  }
  fold_acc[k] <- max(accss)
}
# calculate the mean of the max accuracies for the folds
acccuracy_train <- mean(train_acc)
acccuracy_cv <- mean(fold_acc)
# outputs
train_acc; acccuracy_train
fold_acc; acccuracy_cv
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.sel_model <- predict(sel_model,newdata=subset(test,select=seq(1,14)),type='response')
  test.sel_model <- ifelse(test.sel_model > 0.1*k,1,0)
  misClasificError <- mean(test.sel_model != test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Statistical Significant Predictors removing scale loss

```
# Custom model
custom_model <- glm(detection.status~eye.injury+hook.location+fin.damage+reflex.score, family=binomial(
summary(custom_model)
formula(custom_model)
```

AIC = 218.45

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.custom_model <- predict(custom_model,newdata=subset(train,select=seq(1,14)),type='response')
  train.custom_model <- ifelse(train.custom_model > 0.1*k,1,0)
  misClasificError <- mean(train.custom_model != train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Removing air exposure with k-fold cross validation

```
# split the training data into five parts
partition <- 180/5 # =36
folds <- c(1:36, 37:72, 73:108, 109:144, 145:180)

# Cross-Validation
range.fold <- c(1,2,3,4,5)
fold_acc <- c(1,2,3,4,5)
for (k in range.fold){
  train.data <- train[-((36*(k-1)+1):(36*k)),]
  train.model <- glm(detection.status~eye.injury+hook.location+fin.damage+reflex.score,
                     family=binomial(link='logit'),data=train.data)
```

```
  # calculate validation accuracy:

  # check the categorical variables levels in validation data
  # remove the 'new' levels in validation data
  valid.variables <- train[(36*(k-1)+1):(36*k),1:14]
  valid.data <- train[(36*(k-1)+1):(36*k),]
  for (var in variable.names(train.data)){
    train.var <- pull(train.data[var])
    if (is.factor(train.var)){
      train.levels <- unique(train.var)
      valid.var <- pull(valid.data[var])
      valid.levels <- unique(valid.var)
      if (!(all(valid.levels %in% train.levels))){
        # find the new level(s) in valid but not in train
        new.levels <- valid.levels[!(valid.levels %in% train.levels)]
        valid.variables <- valid.variables[which(valid.var %in% train.levels),]
        valid.data <- valid.data[which(valid.var %in% train.levels),]
      }
    }
  }

  range.decision_boundary <- c(0,1,2,3,4,5,6,7,8,9,10)
  accs <- c(0,1,2,3,4,5,6,7,8,9,10)
  for (i in range.decision_boundary){
    # training accuracy for different decision boundaries:
    valid.model <- predict(train.model,newdata=train.data, type='response')
    valid.model <- ifelse(valid.model > 0.1*i,1,0)
    misClasificError <- mean(valid.model != train.data$detection.status)
    accs[i+1] <- 1-misClasificError
  }
  # selection the max accuracy
  train_acc[k] <- max(accs)
  best_bound <- which.max(accs)

  # compute the validation accuracy
  accss <- c(1,2,3)
  for (i in accss){
    valid.model <- predict(train.model,newdata=valid.variables, type='response')
    valid.model <- ifelse(valid.model > 0.1*(best_bound-2+i),1,0)
    misClasificError <- mean(valid.model != valid.data$detection.status)
    accss[i] <- 1-misClasificError
  }
  fold_acc[k] <- max(accss)
}
# calculate the mean of the max accuracies for the folds
acccuracy_train <- mean(train_acc)
acccuracy_cv <- mean(fold_acc)
# outputs
train_acc; acccuracy_train
fold_acc; acccuracy_cv
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
```

```
  # testing accuracy:
  test.custom_model <- predict(custom_model,newdata=subset(test,select=seq(1,14)),type='response')
  test.custom_model <- ifelse(test.custom_model > 0.1*k,1,0)
  misClasificError <- mean(test.custom_model != test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Try Confusion Matrix

For training data

```
threshold <- 0.5

confusion.train <-
    train %>%
    dplyr::mutate(phat = predict(custom_model,newdata=subset(train,select=seq(1,14)),type='response'),
                  prediction = ifelse(phat > threshold,1,0)) %>%
    dplyr::select(detection.status, prediction, phat)

caret::confusionMatrix(factor(confusion.train$prediction), factor(confusion.train$detection.status))
```

```
incorrect.trainpred<-confusion.train$detection.status!=confusion.train$prediction
train.incorrect <- train[which(incorrect.trainpred),]
```

For testing data

```
threshold <- 0.4

confusion.test <-
    test %>%
    dplyr::mutate(phat = predict(custom_model,newdata=subset(test,select=seq(1,14)),type='response'),
                  prediction = ifelse(phat > threshold,1,0)) %>%
    dplyr::select(detection.status, prediction, phat)

caret::confusionMatrix(factor(confusion.test$prediction), factor(confusion.test$detection.status))
```

```
incorrect.testpred<-confusion.test$detection.status!=confusion.test$prediction
test.incorrect <- test[which(incorrect.testpred),]
```

Lasso

```
# Lasso
x <- model.matrix(detection.status~., train)[,-1];
y <- ifelse(train$detection.status == "1", 1, 0);
cv.lasso <- cv.glmnet(x, y, alpha = 1, family = "binomial"(link='logit'))
lasso.model <- glmnet(x, y, alpha = 1, family = "binomial"(link='logit'), lambda = cv.lasso$lambda.min)
coef(lasso.model)
summary(lasso.model)
```

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  x.train <- model.matrix(detection.status ~., train)[,-1]
```

```
  probabilities <- lasso.model %>% predict(newx = x.train)
  predicted.classes <- ifelse(probabilities > 0.1*k, 1, 0)
  observed.classes <- train$detection.status
  misClasificError <- mean(predicted.classes != observed.classes)
  print(paste('Accuracy',1-misClasificError))
}
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  x.test <- model.matrix(detection.status ~., test)[,-1]
  probabilities <- lasso.model %>% predict(newx = x.test)
  predicted.classes <- ifelse(probabilities > 0.1*k, 1, 0)
  observed.classes <- test$detection.status
  misClasificError <- mean(predicted.classes != observed.classes)
  print(paste('Accuracy',1-misClasificError))
}
```

Subset Selection (Best Subset)

```
# Best Subset Selection
regfit.full = regsubsets(detection.status ~ ., data = train, nvmax = 31)
(reg.summary = summary(regfit.full))
```

```
# Plot RSS, adjusted r-square, Cp, BIC for all the models at once
par(mfrow = c(2, 2))
# RSS Plot
plot(reg.summary$rss, xlab = "Number of Variables", ylab = "RSS", type = "l")
# Adjusted RSq plot
plot(reg.summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted RSq", type = "l")
# Cp plot
plot(reg.summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
# BIC plot
plot(reg.summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")

which.min(reg.summary$rss)
which.max(reg.summary$adjr2)
which.min(reg.summary$cp)
which.min(reg.summary$bic)
```

Subset Selection (Forward)

```
# Forward Selection
regfit.forward = regsubsets(detection.status ~ ., data = train, nvmax = 31, method = "forward")
(forward.summary = summary(regfit.forward))
```

```
# Plot RSS, adjusted r-square, Cp, BIC for all the models at once
par(mfrow = c(2, 2))
# RSS Plot
plot(forward.summary$rss, xlab = "Number of Variables", ylab = "RSS", type = "l")
# Adjusted RSq plot
plot(forward.summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted RSq", type = "l")
# Cp plot
```

```
plot(forward.summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
# BIC plot
plot(forward.summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")

which.min(forward.summary$rss)
which.max(forward.summary$adjr2)
which.min(forward.summary$cp)
which.min(forward.summary$bic)
```

Seems not work. still use AIC (or test accuracy)

Try BIC model selection

```
# Custom model
bic_model <- bic.glm(detection.status~., glm.family=binomial(link='logit'),data=train);
summary(bic_model)
formula(bic_model)
```

Seems not working

Random Forest (with cross-validation)

Random Forest with in-built function

```
# Random Forest with in-built function
trControl <- trainControl(method = "cv", number = 5)
#no_hook_3 <- train[-136,]
glm_default <- train(detection.status~eye.injury+hook.location+fin.damage+air.exposure+reflex.score, me
# Print the results
print(glm_default)
```

Random Forest with self-designed 5-fold cross-validation

```
# split the training data into five parts
partition <- 180/5 # =36
folds <- c(1:36, 37:72, 73:108, 109:144, 145:180)

# Cross-Validation
range.fold <- c(1,2,3,4,5)
fold_acc <- c(1,2,3,4,5)
for (k in range.fold){
  train.data <- train[-((36*(k-1)+1):(36*k)),]
  train.model <- randomForest(detection.status~., data = train.data, ntree = 1000, mtry = 2, importance

  # calculate validation accuracy:

  # check the categorical variables levels in validation data
  # remove the 'new' levels in validation data
  valid.variables <- train[(36*(k-1)+1):(36*k),1:14]
  valid.data <- train[(36*(k-1)+1):(36*k),]
  for (var in variable.names(train.data)){
    train.var <- pull(train.data[var])
    if (is.factor(train.var)){
      train.levels <- unique(train.var)
```

```
      valid.var <- pull(valid.data[var])
      valid.levels <- unique(valid.var)
      if (!(all(valid.levels %in% train.levels))){
        # find the new level(s) in valid but not in train
        new.levels <- valid.levels[!(valid.levels %in% train.levels)]
        valid.variables <- valid.variables[which(valid.var %in% train.levels),]
        valid.data <- valid.data[which(valid.var %in% train.levels),]
      }
    }
  }

  # accuracy for one fold
  rf_pred <- predict(train.model, valid.data, type = "class")
  fold_acc[k] <- mean(rf_pred == valid.data$detection.status)
}
# calculate the mean of the max accuracies for the folds
acccuracy_cv <- mean(fold_acc)
# outputs
fold_acc; acccuracy_cv
```

Apply random forest model to test data

```
rf_model <- randomForest(detection.status~eye.injury+hook.location+fin.damage+air.exposure+reflex.score
rf_model
```

```
rftrain_pred <- predict(rf_model, train, type = "class")
mean(rftrain_pred == train$detection.status)
table(rftrain_pred, train$detection.status)
```

The performance on training data are much better than all other models previously.

```
rf_pred <- predict(rf_model, test, type = "class")
mean(rf_pred == test$detection.status)
table(rf_pred, test$detection.status)
```

The full model overfits.

Logistic Model Trees

```
trControl <- trainControl(method = "cv", number = 5)
#no_hook_3 <- train[-136,]
glm_default <- train(detection.status~eye.injury+hook.location+fin.damage+air.exposure+reflex.score, me
# Print the results
print(glm_default)
```

Apply LMT to test data

```
lmt_model <- LMT(detection.status~eye.injury+hook.location+fin.damage+air.exposure+reflex.score, data =
summary(lmt_model)
```

```
lmttrain_pred <- predict(lmt_model, train, type = "class")
mean(lmttrain_pred == train$detection.status)
table(lmttrain_pred, train$detection.status)
```

```
lmt_pred <- predict(lmt_model, test, type = "class")
mean(lmt_pred == test$detection.status)
table(lmt_pred, test$detection.status)
```

Full model and custom model give the same results

RRF (Regularized Random Forest)

RRF with cross-validation

```
trControl <- trainControl(method = "cv", number = 5, search = "grid")

rrf_default <- train(detection.status~., data = train, method = "RRF", metric = "Accuracy", trControl =
# Print the results
print(rrf_default)
```

Apply RRF to test data

```
rrf_model <- RRF(detection.status ~ eye.injury+hook.location+fin.damage+air.exposure+reflex.score, trai
rrf_model
```

```
rrftrain_pred <- predict(rrf_model, train, type = "class")
mean(rrftrain_pred == train$detection.status)
table(rrftrain_pred, train$detection.status)
```

```
rrf_pred <- predict(rrf_model, test, type = "class")
mean(rrf_pred == test$detection.status)
table(rrf_pred, test$detection.status)
```

Again, the full model overfits.

KNN

```
trControl <- trainControl(method = "cv", number = 5)
knn_default <- train(detection.status~., data = train, method = "knn", metric = "Accuracy", trControl =
# Print the results
print(knn_default)
```

Need to convert factors (with various levels) into numerical data, and further standardize. Not actually suitable.

NNET

```
trControl <- trainControl(method = "cv", number = 5, search = "grid")
knn_default <- train(detection.status~eye.injury+hook.location+fin.damage+air.exposure+reflex.score, da
# Print the results
print(knn_default)
```

Seems not working.

Tuning parameters for RF, RRF

```
rf_model <- randomForest(detection.status~eye.injury+hook.location+fin.damage+air.exposure+reflex.score
rf_model
```

```
rftrain_pred <- predict(rf_model, train, type = "class")
mean(rftrain_pred == train$detection.status)
table(rftrain_pred, train$detection.status)
```

```
rf_pred <- predict(rf_model, test, type = "class")
mean(rf_pred == test$detection.status)
table(rf_pred, test$detection.status)
```

```
mtry.options = c(1,2,3,4,5)
train.accuracy = c(1,2,3,4,5)
test.accuracy = c(1,2,3,4,5)
for (mtry in mtry.options){
  rf_model <- randomForest(detection.status~eye.injury+hook.location+fin.damage+air.exposure+reflex.sco
  # train accuracy
  rftrain_pred <- predict(rf_model, train, type = "class")
  train.accuracy[mtry] <- mean(rftrain_pred == train$detection.status)
  # test accuracy
  rf_pred <- predict(rf_model, test, type = "class")
  test.accuracy[mtry] <- mean(rf_pred == test$detection.status)
}
```

```
plot(mtry.options, train.accuracy, col="blue", type="o", ylim=range(c(train.accuracy,test.accuracy)))
points(mtry.options, test.accuracy, col="red", pch="*")
lines(mtry.options, test.accuracy, col="red")
```

```
ntree.options = c(1,2,3,4,5,6,7,8,9,10)
train.accuracy = c(1,2,3,4,5,6,7,8,9,10)
test.accuracy = c(1,2,3,4,5,6,7,8,9,10)
for (ntree in ntree.options){
  rf_model <- randomForest(detection.status~eye.injury+reflex.score+hook.location+fin.damage+air.exposu
  # train accuracy
  rftrain_pred <- predict(rf_model, train, type = "class")
  train.accuracy[ntree] <- mean(rftrain_pred == train$detection.status)
  # test accuracy
  rf_pred <- predict(rf_model, test, type = "class")
  test.accuracy[ntree] <- mean(rf_pred == test$detection.status)
}
```

```
plot(ntree.options, train.accuracy, col="blue", type="o", ylim=range(c(train.accuracy,test.accuracy)))
points(ntree.options, test.accuracy, col="red", pch="*")
lines(ntree.options, test.accuracy, col="red")
```

```
mtry.options = c(1,2,3,4,5,6)
train.accuracy = c(1,2,3,4,5,6)
test.accuracy = c(1,2,3,4,5,6)
for (mtry in mtry.options){
  # detection.status~eye.injury+hook.location+fin.damage+reflex.score+air.exposure
  rf_model <- randomForest(detection.status~eye.injury+reflex.score+hook.location+fin.damage+air.exposu
  # train accuracy
  rftrain_pred <- predict(rf_model, train, type = "class")
  train.accuracy[mtry] <- mean(rftrain_pred == train$detection.status)
  # test accuracy
  rf_pred <- predict(rf_model, test, type = "class")
  test.accuracy[mtry] <- mean(rf_pred == test$detection.status)
}
```

```
plot(mtry.options, train.accuracy, col="blue", type="o", ylim=range(c(train.accuracy,test.accuracy)))
points(mtry.options, test.accuracy, col="red", pch="*")
lines(mtry.options, test.accuracy, col="red")
```

Analyze the updated data with condensed scores or separate variables

Condense 5 injury predictor variables to injury score

Read data

```
# read data and perspective
newraw_data <- read_xlsx("updated_data_sepearate_variables.xlsx", sheet = "data", col_names = TRUE)
head(newraw_data); dim(newraw_data)
```

Some columns incorrectly converted to char() due to NA

```
# correct the variables data type
# Warning: note that NAs introduced by coercion
newraw_data <- newraw_data %>%
               mutate(air.exposure = as.double(air.exposure)) %>%
               mutate(hook.location = as.double(hook.location)) %>%
               mutate(mean.fat = as.double(mean.fat))
head(newraw_data); dim(newraw_data)
```

Removing all the NA's and Unknown's as missing data (40 in total)

```
# remove the missing data
new_complete_data <- newraw_data %>%
                   filter(!is.na(air.exposure) & !is.na(hook.location) & !is.na(mean.fat) &
                          (tolower(population) != "unknown") & (tolower(sex) != "unknown") )
head(new_complete_data); dim(new_complete_data)
```

Refine the data table (remove fish.no which is equivalent to tag.id)

```
# remove fish.no column and check that tag.id is an unique identifier
new_complete_data <- new_complete_data %>%
                     select(-"fish.no")
all(new_complete_data$tag.id == unique(new_complete_data$tag.id))
head(new_complete_data); dim(new_complete_data)
```

Change the date to year

```
new_complete_data <- new_complete_data %>%
                     mutate(year=substr(date,1,4), .after=date)
head(new_complete_data); dim(new_complete_data)
```

Remove fin damage, wound score, scale loss, eye injury, and bleeding, VOR, tail.flex, body.flex, venting, orientation to analyze injury.score

```
# remove fish.no column and check that tag.id is an unique identifier
new_complete_data <- new_complete_data %>%
                     select(-"fin.damage") %>% select(-"wound.score") %>% select(-"scale.loss") %>%
```

```
                      select(-"eye.injury") %>% select(-"bleed") %>% select(-"VOR") %>%
                      select(-"tail.flex") %>% select(-"body.flex") %>% select(-"venting") %>%
                      select(-"orientation")
all(new_complete_data$tag.id == unique(new_complete_data$tag.id))
head(new_complete_data); dim(new_complete_data)
```

```
# For injury.score treat as categorical variable:
vs.injury <- table(new_complete_data$injury.score, new_complete_data$detection.status)
names(dimnames(vs.injury)) <- c("injury.score", "detection.status")
addmargins(vs.injury)

# data visualization:
injury_table <- table(new_complete_data$detection.status, new_complete_data$injury.score)
barplot(injury_table, xlab="injury.score", col=c("grey", "light blue"), legend = rownames(injury_table)

# data visualization with fractions
bar_plot <- function(injury_table){
  ggplot(as.data.frame(injury_table),aes(x=Var2,y=Freq,fill=Var1)) +
      geom_bar(stat="identity",position="stack")+
      xlab("injury.score") + ylab("freq") + labs(fill="detection.status") +
      geom_text(aes(label=fraction(Freq)),position="stack",vjust=1)+
      scale_fill_manual(values=c("grey","light blue"))+
      theme_bw()
}
bar_plot(injury_table)
```

Model fitting

Factor the categorical variables

```
## Factor
new_fit_dataset<-new_complete_data %>%
                  mutate(year=as.factor(year)) %>%
                  mutate(population=as.factor(population)) %>%
                  mutate(sex=as.factor(sex)) %>%
                  mutate(origin=as.factor(origin)) %>%
                  mutate(hook.location=as.factor(hook.location)) %>%
                  mutate(injury.score=as.factor(injury.score)) %>%
                  mutate(detection.status=as.factor(detection.status)) %>%
                  #mutate(air.exposure=as.factor(air.exposure)) %>%
                  select(-"tag.id") %>%
                  select(-"date")

str(new_fit_dataset)
```

Split into training dataset and testing dataset (4:1) proportional to detection.status

```
# Sample from detected:

new_detected <- new_fit_dataset %>% filter(detection.status == 1)
smp_size <- floor(0.75 * nrow(new_detected))
set.seed(450)
new_detected_train <- sample(seq_len(nrow(new_detected)), size = smp_size)
```

```
new_train_det <- new_detected[new_detected_train, ]
new_test_det <- new_detected[-new_detected_train, ]

# Sample from notdetected:
new_notdetected <- new_fit_dataset %>% filter(detection.status == 0)
smp_size <- floor(0.75 * nrow(new_notdetected))
set.seed(451)
new_notdetected_train <- sample(seq_len(nrow(new_notdetected)), size = smp_size)

new_train_notdet <- new_notdetected[new_notdetected_train, ]
new_test_notdet <- new_notdetected[-new_notdetected_train, ]

# Combine detected and notdetected
new_train <- rbind(new_train_det, new_train_notdet)
new_test <- rbind(new_test_det, new_test_notdet)

# Shuffle the rows
#set.seed(452)
new_rows_train <- sample(nrow(new_train))
new_train <- new_train[new_rows_train, ]

#set.seed(453)
new_rows_test <- sample(nrow(new_test))
new_test <- new_test[new_rows_test, ]
```

Full model

```
# The full model
new_fullmodel <- glm(detection.status~.,family=binomial(link='logit'),data=new_train)
summary(new_fullmodel)
```

AIC = 227.49

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.new_fullmodel <- predict(new_fullmodel,newdata=subset(new_train,select=seq(1,10)),type='respons
  train.new_fullmodel <- ifelse(train.new_fullmodel > 0.1*k,1,0)
  misClasificError <- mean(train.new_fullmodel != new_train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.new_fullmodel <- predict(new_fullmodel,newdata=subset(new_test,select=seq(1,10)),type='response'
  test.new_fullmodel <- ifelse(test.new_fullmodel > 0.1*k,1,0)
  misClasificError <- mean(test.new_fullmodel != new_test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Backward Selection

```
# Backward Selection
new_backwards = stats::step(new_fullmodel)
summary(new_backwards)
formula(new_backwards)
```

AIC = 213.02

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.new_backmodel <- predict(new_backwards,newdata=subset(new_train,select=seq(1,10)),type='respons
  train.new_backmodel <- ifelse(train.new_backmodel > 0.1*k,1,0)
  misClasificError <- mean(train.new_backmodel != new_train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.new_backmodel <- predict(new_backwards,newdata=subset(new_test,select=seq(1,10)),type='response'
  test.new_backmodel <- ifelse(test.new_backmodel > 0.1*k,1,0)
  misClasificError <- mean(test.new_backmodel != new_test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Forward Selection

```
# Forward Selection
new_basemodel <- glm(detection.status~NULL, family=binomial(link='logit'),data=new_train);
new_forwards <- stats::step(new_basemodel,scope=list(lower=formula(new_basemodel),upper=formula(new_full
summary(new_forwards)
formula(new_forwards)
```

AIC = 213.02

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.foremodel <- predict(new_forwards,newdata=subset(new_train,select=seq(1,10)),type='response')
  train.foremodel <- ifelse(train.foremodel > 0.1*k,1,0)
  misClasificError <- mean(train.foremodel != new_train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.new_foremodel <- predict(new_forwards,newdata=subset(new_test,select=seq(1,10)),type='response')
  test.new_foremodel <- ifelse(test.new_foremodel > 0.1*k,1,0)
  misClasificError <- mean(test.new_foremodel != new_test$detection.status)
```

```
  print(paste('Accuracy',1-misClasificError))
}
```

Custom Model

```
# Custom model
new_custom_model <- glm(detection.status~hook.location+injury.score+reflex.score, family=binomial(link=
summary(new_custom_model)
formula(new_custom_model)
```

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.new_custom_model <- predict(new_custom_model,newdata=subset(new_train,select=seq(1,10)),type='r
  train.new_custom_model <- ifelse(train.new_custom_model > 0.1*k,1,0)
  misClasificError <- mean(train.new_custom_model != new_train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.new_custom_model <- predict(new_custom_model,newdata=subset(new_test,select=seq(1,10)),type='resp
  test.new_custom_model <- ifelse(test.new_custom_model > 0.1*k,1,0)
  misClasificError <- mean(test.new_custom_model != new_test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Custom model removing air exposure with k-fold cross validation

```
# split the training data into five parts
partition <- 180/5 # =36
folds <- c(1:36, 37:72, 73:108, 109:144, 145:180)

# Cross-Validation
range.fold <- c(1,2,3,4,5)
fold_acc <- c(1,2,3,4,5)
for (k in range.fold){
  train.data <- new_train[-((36*(k-1)+1):(36*k)),]
  train.model <- glm(detection.status~hook.location+injury.score+reflex.score,
                     family=binomial(link='logit'),data=train.data)

  # calculate validation accuracy:

  # check the categorical variables levels in validation data
  # remove the 'new' levels in validation data
  valid.variables <- new_train[(36*(k-1)+1):(36*k),1:10]
  valid.data <- new_train[(36*(k-1)+1):(36*k),]
  for (var in variable.names(train.data)){
    train.var <- pull(train.data[var])
    if (is.factor(train.var)){
      train.levels <- unique(train.var)
```

```
      valid.var <- pull(valid.data[var])
      valid.levels <- unique(valid.var)
      if (!(all(valid.levels %in% train.levels))){
        # find the new level(s) in valid but not in train
        new.levels <- valid.levels[!(valid.levels %in% train.levels)]
        valid.variables <- valid.variables[which(valid.var %in% train.levels),]
        valid.data <- valid.data[which(valid.var %in% train.levels),]
      }
    }
  }

  range.decision_boundary <- c(0,1,2,3,4,5,6,7,8,9,10)
  accs <- c(0,1,2,3,4,5,6,7,8,9,10)
  for (i in range.decision_boundary){
    # training accuracy for different decision boundaries:
    valid.model <- predict(train.model,newdata=train.data, type='response')
    valid.model <- ifelse(valid.model > 0.1*i,1,0)
    misClasificError <- mean(valid.model != train.data$detection.status)
    accs[i+1] <- 1-misClasificError
  }
  # selection the max accuracy
  train_acc[k] <- max(accs)
  best_bound <- which.max(accs)

  # compute the validation accuracy
  accss <- c(1,2,3)
  for (i in accss){
    valid.model <- predict(train.model,newdata=valid.variables, type='response')
    valid.model <- ifelse(valid.model > 0.1*(best_bound-2+i),1,0)
    misClasificError <- mean(valid.model != valid.data$detection.status)
    accss[i] <- 1-misClasificError
  }
  fold_acc[k] <- max(accss)
}
# calculate the mean of the max accuracies for the folds
acccuracy_train <- mean(train_acc)
acccuracy_cv <- mean(fold_acc)
# outputs
train_acc; acccuracy_train
fold_acc; acccuracy_cv
```

Followed the same process for new data with injury.score, not as good as the separate variables

Expand the reflex score to 5 separate reflex predictors

Read data

```
# read data and perspective
newraw_data <- read_xlsx("updated_data_sepearate_variables.xlsx", sheet = "data", col_names = TRUE)
head(newraw_data); dim(newraw_data)
```

Some columns incorrectly converted to char() due to NA

```
# correct the variables data type
# Warning: note that NAs introduced by coercion
newraw_data <- newraw_data %>%
               mutate(air.exposure = as.double(air.exposure)) %>%
               mutate(hook.location = as.double(hook.location)) %>%
               mutate(mean.fat = as.double(mean.fat))
head(newraw_data); dim(newraw_data)
```

Removing all the NA's and Unknown's as missing data (40 in total)

```
# remove the missing data
new_complete_data <- newraw_data %>%
                     filter(!is.na(air.exposure) & !is.na(hook.location) & !is.na(mean.fat) &
                            (tolower(population) != "unknown") & (tolower(sex) != "unknown") )
head(new_complete_data); dim(new_complete_data)
```

Refine the data table (remove fish.no which is equivalent to tag.id)

```
# remove fish.no column and check that tag.id is an unique identifier
new_complete_data <- new_complete_data %>%
                      select(-"fish.no")
all(new_complete_data$tag.id == unique(new_complete_data$tag.id))
head(new_complete_data); dim(new_complete_data)
```

Change the date to year

```
new_complete_data <- new_complete_data %>%
                      mutate(year=substr(date,1,4), .after=date)
head(new_complete_data); dim(new_complete_data)
```

Remove injury.score and reflex.score

```
# remove fish.no column and check that tag.id is an unique identifier
new_complete_data <- new_complete_data %>%
                      select(-"injury.score") %>% select(-"reflex.score")
all(new_complete_data$tag.id == unique(new_complete_data$tag.id))
head(new_complete_data); dim(new_complete_data)
```

```
# For VOR treat as categorical variable:
vs.vor <- table(new_complete_data$VOR, new_complete_data$detection.status)
names(dimnames(vs.vor)) <- c("VOR", "detection.status")
addmargins(vs.vor)

# data visualization:
vor_table <- table(new_complete_data$detection.status, new_complete_data$VOR)
barplot(vor_table, xlab="VOR", col=c("grey", "light blue"), legend = rownames(vor_table), beside=TRUE)

# data visualization with fractions
bar_plot <- function(vor_table){
  ggplot(as.data.frame(vor_table),aes(x=Var2,y=Freq,fill=Var1)) +
      geom_bar(stat="identity",position="stack")+
```

```
      xlab("VOR") + ylab("freq") + labs(fill="detection.status") +
      geom_text(aes(label=fraction(Freq)),position="stack",vjust=1)+
      scale_fill_manual(values=c("grey","light blue"))+
      theme_bw()
}
bar_plot(vor_table)
```

```
# For VOR treat as categorical variable:
vs.tail <- table(new_complete_data$tail.flex, new_complete_data$detection.status)
names(dimnames(vs.tail)) <- c("tail.flex", "detection.status")
addmargins(vs.tail)

# data visualization:
tail_table <- table(new_complete_data$detection.status, new_complete_data$tail.flex)
barplot(tail_table, xlab="tail.flex", col=c("grey", "light blue"), legend = rownames(tail_table), beside

# data visualization with fractions
bar_plot <- function(tail_table){
  ggplot(as.data.frame(tail_table),aes(x=Var2,y=Freq,fill=Var1)) +
      geom_bar(stat="identity",position="stack")+
      xlab("tail.flex") + ylab("freq") + labs(fill="detection.status") +
      geom_text(aes(label=fraction(Freq)),position="stack",vjust=1)+
      scale_fill_manual(values=c("grey","light blue"))+
      theme_bw()
}
bar_plot(tail_table)
```

```
# For VOR treat as categorical variable:
vs.body <- table(new_complete_data$body.flex, new_complete_data$detection.status)
names(dimnames(vs.body)) <- c("body.flex", "detection.status")
addmargins(vs.body)

# data visualization:
body_table <- table(new_complete_data$detection.status, new_complete_data$body.flex)
barplot(body_table, xlab="body.flex", col=c("grey", "light blue"), legend = rownames(body_table), beside

# data visualization with fractions
bar_plot <- function(body_table){
  ggplot(as.data.frame(body_table),aes(x=Var2,y=Freq,fill=Var1)) +
      geom_bar(stat="identity",position="stack")+
      xlab("body.flex") + ylab("freq") + labs(fill="detection.status") +
      geom_text(aes(label=fraction(Freq)),position="stack",vjust=1)+
      scale_fill_manual(values=c("grey","light blue"))+
      theme_bw()
}
bar_plot(body_table)
```

```
# For VOR treat as categorical variable:
vs.venting <- table(new_complete_data$venting, new_complete_data$detection.status)
names(dimnames(vs.venting)) <- c("venting", "detection.status")
addmargins(vs.venting)

# data visualization:
venting_table <- table(new_complete_data$detection.status, new_complete_data$venting)
```

```
barplot(venting_table, xlab="venting", col=c("grey", "light blue"), legend = rownames(venting_table), be

# data visualization with fractions
bar_plot <- function(venting_table){
  ggplot(as.data.frame(venting_table),aes(x=Var2,y=Freq,fill=Var1)) +
      geom_bar(stat="identity",position="stack")+
      xlab("venting") + ylab("freq") + labs(fill="detection.status") +
      geom_text(aes(label=fraction(Freq)),position="stack",vjust=1)+
      scale_fill_manual(values=c("grey","light blue"))+
      theme_bw()
}
bar_plot(venting_table)
```

```
# For VOR treat as categorical variable:
vs.orientation <- table(new_complete_data$orientation, new_complete_data$detection.status)
names(dimnames(vs.orientation)) <- c("orientation", "detection.status")
addmargins(vs.orientation)

# data visualization:
orientation_table <- table(new_complete_data$detection.status, new_complete_data$orientation)
barplot(orientation_table, xlab="orientation", col=c("grey", "light blue"), legend = rownames(vor_table)

# data visualization with fractions
bar_plot <- function(orientation_table){
  ggplot(as.data.frame(orientation_table),aes(x=Var2,y=Freq,fill=Var1)) +
      geom_bar(stat="identity",position="stack")+
      xlab("orientation") + ylab("freq") + labs(fill="detection.status") +
      geom_text(aes(label=fraction(Freq)),position="stack",vjust=1)+
      scale_fill_manual(values=c("grey","light blue"))+
      theme_bw()
}
bar_plot(orientation_table)
```

Model fitting

Factor the categorical variables

```
## Factor
new_fit_dataset<-new_complete_data %>%
                  mutate(year=as.factor(year)) %>%
                  mutate(population=as.factor(population)) %>%
                  mutate(sex=as.factor(sex)) %>%
                  mutate(origin=as.factor(origin)) %>%
                  mutate(hook.location=as.factor(hook.location)) %>%
                  mutate(fin.damage=as.factor(fin.damage)) %>%
                  mutate(wound.score=as.factor(wound.score)) %>%
                  mutate(scale.loss=as.factor(scale.loss)) %>%
                  mutate(eye.injury=as.factor(eye.injury)) %>%
                  mutate(bleed=as.factor(bleed)) %>%
                  mutate(VOR=as.factor(VOR)) %>%
                  mutate(tail.flex=as.factor(tail.flex)) %>%
                  mutate(body.flex=as.factor(body.flex)) %>%
                  mutate(venting=as.factor(venting)) %>%
                  mutate(orientation=as.factor(orientation)) %>%
```

```
                  mutate(detection.status=as.factor(detection.status)) %>%
                  #mutate(air.exposure=as.factor(air.exposure)) %>%
                  select(-"tag.id") %>%
                  select(-"date")

str(new_fit_dataset)
```

Standardize

```
new_fit_dataset$air.exposure <- scale(new_fit_dataset$air.exposure)
str(new_fit_dataset)
```

Split into training dataset and testing dataset (4:1) proportional to detection.status

```
# Sample from detected:

new_detected <- new_fit_dataset %>% filter(detection.status == 1)
smp_size <- floor(0.75 * nrow(new_detected))
set.seed(450)
new_detected_train <- sample(seq_len(nrow(new_detected)), size = smp_size)

new_train_det <- new_detected[new_detected_train, ]
new_test_det <- new_detected[-new_detected_train, ]

# Sample from notdetected:
new_notdetected <- new_fit_dataset %>% filter(detection.status == 0)
smp_size <- floor(0.75 * nrow(new_notdetected))
set.seed(451)
new_notdetected_train <- sample(seq_len(nrow(new_notdetected)), size = smp_size)

new_train_notdet <- new_notdetected[new_notdetected_train, ]
new_test_notdet <- new_notdetected[-new_notdetected_train, ]

# Combine detected and notdetected
new_train <- rbind(new_train_det, new_train_notdet)
new_test <- rbind(new_test_det, new_test_notdet)

# Shuffle the rows
#set.seed(452)
new_rows_train <- sample(nrow(new_train))
new_train <- new_train[new_rows_train, ]

#set.seed(453)
new_rows_test <- sample(nrow(new_test))
new_test <- new_test[new_rows_test, ]
```

Full model

```
# The full model
new_fullmodel <- glm(detection.status~.,family=binomial(link='logit'),data=new_train)
summary(new_fullmodel)
```

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.new_fullmodel <- predict(new_fullmodel,newdata=subset(new_train,select=seq(1,18)),type='response
  train.new_fullmodel <- ifelse(train.new_fullmodel > 0.1*k,1,0)
  misClasificError <- mean(train.new_fullmodel != new_train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.new_fullmodel <- predict(new_fullmodel,newdata=subset(new_test,select=seq(1,18)),type='response'
  test.new_fullmodel <- ifelse(test.new_fullmodel > 0.1*k,1,0)
  misClasificError <- mean(test.new_fullmodel != new_test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Backward Selection

```
# Backward Selection
new_backwards = stats::step(new_fullmodel)
summary(new_backwards)
formula(new_backwards)
```

```
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.new_backmodel <- predict(new_backwards,newdata=subset(new_train,select=seq(1,18)),type='respons
  train.new_backmodel <- ifelse(train.new_backmodel > 0.1*k,1,0)
  misClasificError <- mean(train.new_backmodel != new_train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

```
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.new_backmodel <- predict(new_backwards,newdata=subset(new_test,select=seq(1,18)),type='response'
  test.new_backmodel <- ifelse(test.new_backmodel > 0.1*k,1,0)
  misClasificError <- mean(test.new_backmodel != new_test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Forward Selection

```
# Forward Selection
new_basemodel <- glm(detection.status~NULL, family=binomial(link='logit'),data=new_train);
new_forwards <- stats::step(new_basemodel,scope=list(lower=formula(new_basemodel),upper=formula(new_full
summary(new_forwards)
formula(new_forwards)
```

```r
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.foremodel <- predict(new_forwards,newdata=subset(new_train,select=seq(1,18)),type='response')
  train.foremodel <- ifelse(train.foremodel > 0.1*k,1,0)
  misClasificError <- mean(train.foremodel != new_train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

```r
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.new_foremodel <- predict(new_forwards,newdata=subset(new_test,select=seq(1,18)),type='response')
  test.new_foremodel <- ifelse(test.new_foremodel > 0.1*k,1,0)
  misClasificError <- mean(test.new_foremodel != new_test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Custom Model

```r
# Custom model
new_custom_model <- glm(detection.status~hook.location+eye.injury+fin.damage+orientation, family=binomia
summary(new_custom_model)
formula(new_custom_model)
```

```r
# training accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # training accuracy:
  train.new_custom_model <- predict(new_custom_model,newdata=subset(new_train,select=seq(1,18)),type='r
  train.new_custom_model <- ifelse(train.new_custom_model > 0.1*k,1,0)
  misClasificError <- mean(train.new_custom_model != new_train$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

```r
# testing accuracy:
range <- c(0,1,2,3,4,5,6,7,8,9,10)
for (k in range){
  # testing accuracy:
  test.new_custom_model <- predict(new_custom_model,newdata=subset(new_test,select=seq(1,18)),type='resp
  test.new_custom_model <- ifelse(test.new_custom_model > 0.1*k,1,0)
  misClasificError <- mean(test.new_custom_model != new_test$detection.status)
  print(paste('Accuracy',1-misClasificError))
}
```

Custom model removing air exposure with k-fold cross validation

```r
# split the training data into five parts
partition <- 180/5 # =36
folds <- c(1:36, 37:72, 73:108, 109:144, 145:180)

# Cross-Validation
```

```
range.fold <- c(1,2,3,4,5)
fold_acc <- c(1,2,3,4,5)
for (k in range.fold){
  train.data <- new_train[-((36*(k-1)+1):(36*k)),]
  train.model <- glm(detection.status~hook.location+eye.injury+fin.damage+orientation,
                     family=binomial(link='logit'),data=train.data)

  # calculate validation accuracy:

  # check the categorical variables levels in validation data
  # remove the 'new' levels in validation data
  valid.variables <- new_train[(36*(k-1)+1):(36*k),1:18]
  valid.data <- new_train[(36*(k-1)+1):(36*k),]
  for (var in variable.names(train.data)){
    train.var <- pull(train.data[var])
    if (is.factor(train.var)){
      train.levels <- unique(train.var)
      valid.var <- pull(valid.data[var])
      valid.levels <- unique(valid.var)
      if (!(all(valid.levels %in% train.levels))){
        # find the new level(s) in valid but not in train
        new.levels <- valid.levels[!(valid.levels %in% train.levels)]
        valid.variables <- valid.variables[which(valid.var %in% train.levels),]
        valid.data <- valid.data[which(valid.var %in% train.levels),]
      }
    }
  }

  range.decision_boundary <- c(0,1,2,3,4,5,6,7,8,9,10)
  accs <- c(0,1,2,3,4,5,6,7,8,9,10)
  for (i in range.decision_boundary){
    # training accuracy for different decision boundaries:
    valid.model <- predict(train.model,newdata=train.data, type='response')
    valid.model <- ifelse(valid.model > 0.1*i,1,0)
    misClasificError <- mean(valid.model != train.data$detection.status)
    accs[i+1] <- 1-misClasificError
  }
  # selection the max accuracy
  train_acc[k] <- max(accs)
  best_bound <- which.max(accs)

  # compute the validation accuracy
  accss <- c(1,2,3)
  for (i in accss){
    valid.model <- predict(train.model,newdata=valid.variables, type='response')
    valid.model <- ifelse(valid.model > 0.1*(best_bound-3+i),1,0)
    misClasificError <- mean(valid.model != valid.data$detection.status)
    accss[i] <- 1-misClasificError
  }
  fold_acc[k] <- max(accss)
}
# calculate the mean of the max accuracies for the folds
acccuracy_train <- mean(train_acc)
```

```r
acccuracy_cv <- mean(fold_acc)
# outputs
train_acc; acccuracy_train
fold_acc; acccuracy_cv
```

Followed the same process for new data with expanded reflex score (separate predictors), it performs slightly better.