Predicting Paper Breaks in Continuous Manufacturing

Arjun Vijay Anup - 24215155

Logistic Regression Model

- The Processminer dataset is loaded, which contains sensor reading (`x1-x59`) and a binary target variable (`break`, `no_break`).

```
# Loading Processminer data
load("data_processminer.RData")
```

- The data is then split into 80% training and 20% test set and standardized all the features using only the training set standardizing scale. This is to prevent data leakage.
- Logistic regression using all the available feature in the dataset is implemented using a **4-fold** cross-validation, repeated **20 times**.
- In-sample predictive performance is evaluated by calculating *Accuracy*, *Sensitivity* and *Specificity* for each fold over 20 replications. This ensures stable performance estimates across data variations.

```
# Setting seed for reproducibility
set.seed(24215155)
# Splitting training and test data
n <- nrow(data_processminer)
test <- sample(1:n, n*0.2)
data.test <- data_processminer[test, ]
train <- setdiff(1:n, test)
data.train <- data_processminer[train, ]
n.train <- nrow(data.train)
# Standardizing train data
x.scale <- scale(data.train[,-1])
data.train[,-1] <- x.scale
# Standardizing test data by using train data scaling
data.test[,-1] <- scale(data.test[,-1],
                        center = attr(x.scale, "scaled:center"),
                        scale = attr(x.scale, "scaled:scale"))
# Setting number of folds and replications
K <- 4
R <- 20
# Storing performance metrics
out <- vector("list", R)
# For each replication
for(r in 1:R){
  # Storing performance metrics for each fold
  metrics <- array(NA, c(K, 1, 3))
  # Randomizing fold data indexes to get random selection of data
  folds <- rep(1:K, ceiling(n.train/K))
  folds <- sample(folds)
  folds <- folds[1:n.train]
  # for each fold
  for (k in 1:K){
    # Setting indexes for training and validation data
    train.fold <- which(folds!=k)
    val.fold <- setdiff(1:n.train, train.fold)
    # Fitting logistic regression model on training data
    fit.log <- suppressWarnings(glm(y ~ ., data = data.train,
                   family = "binomial", subset = train.fold))
    # Predicting classification in validation data fold
```

```r
    pred.log <- predict(fit.log, type = "response",
                        newdata = data.train[val.fold,])
    pred.log <- ifelse(pred.log > 0.5, "break", "no_break")
    pred.log <- factor(pred.log, levels = c("no_break","break"))
    # Evaluating metrics
    tab <- table(data.train$y[val.fold], pred.log)
    acc <- sum(diag(tab))/sum(tab)
    sens <- tab[2,2]/sum(tab[2,])
    spec <- tab[1,1]/sum(tab[1,])
    metrics[k,1,] <- c(acc = acc, sens = sens, spec = spec)
  }
  out[[r]] <- metrics
}
# Aggregate results
avg <- sapply(out, function(x) apply(x, c(2,3), mean))
avg <- array(avg, c(1,3,R))

mean.acc <- rowMeans(avg[,1,, drop = FALSE])
mean.sens <- rowMeans(avg[,2,, drop = FALSE])
mean.spec <- rowMeans(avg[,3,, drop = FALSE])
# Dataframe for performance metrics
perf.log.met<-data.frame(Model = "Logistic", Accuracy = mean.acc,
                         Sensitivity = mean.sens, Specificity = mean.spec)
# Printing table
knitr::kable(perf.log.met, digits = 4,
             caption = "Performance Metrics - Logistic Regression")
```

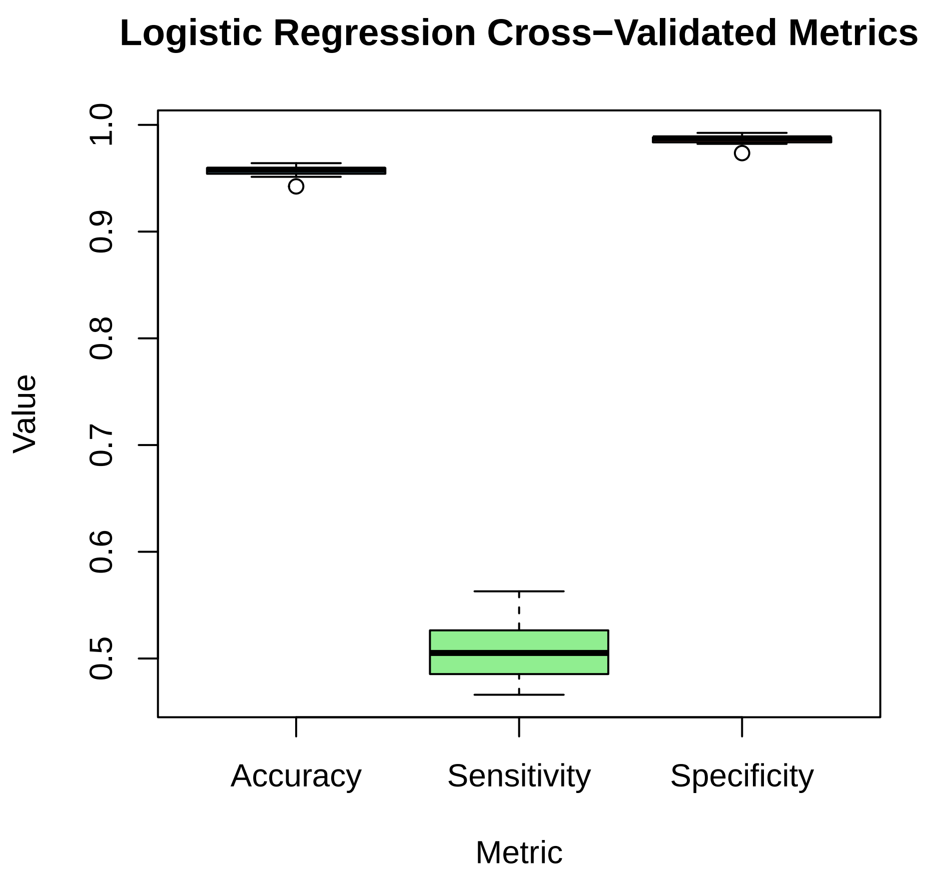
Table 1: Performance Metrics - Logistic Regression

Model	Accuracy	Sensitivity	Specificity
Logistic	0.9565	0.5068	0.9863

```r
# Combining all three metrics into one matrix
box.acc <- c(t(avg[,1,]))  # Accuracy
box.sens <- c(t(avg[,2,])) # Sensitivity
box.spec <- c(t(avg[,3,])) # Specificity

# dataframe for plotting
box.data <- data.frame(
  Metric = rep(c("Accuracy", "Sensitivity", "Specificity"),
               each = length(box.acc)),
  Value = c(box.acc, box.sens, box.spec)
)

# Boxplot for performance metrics
boxplot(Value ~ Metric, data = box.data,
        col = c("skyblue", "lightgreen", "salmon"),
        main = "Logistic Regression Cross-Validated Metrics")
```

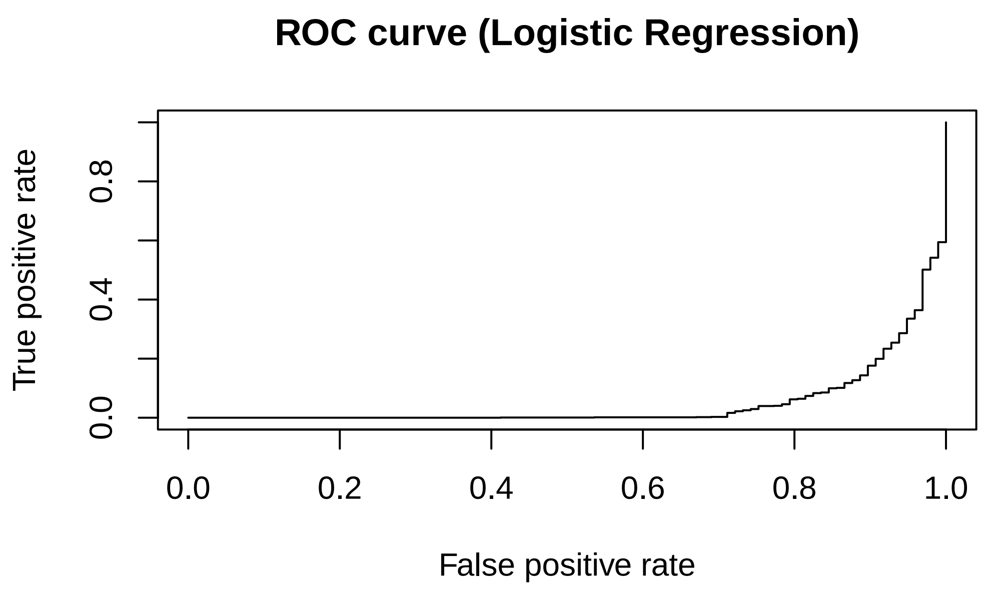


- From the table and the box plots, we observe a high mean accuracy (≈ 95) and mean specificity (≈ 98) but a lower mean sensitivity ≈ 50.
- Sensitivity is relatively low, which denotes that the model misses many `break` occurences. Therefore, the high accuracy value is misleading here due to class imbalance denoted by low sensitivity value.
- As the threshold taken above to train the model was 0.5 to obtain the above performance metric, we apply threshold tuning using ROC, PR and F1 score using the `ROCR` package to improve the model in-sample predictive performance.

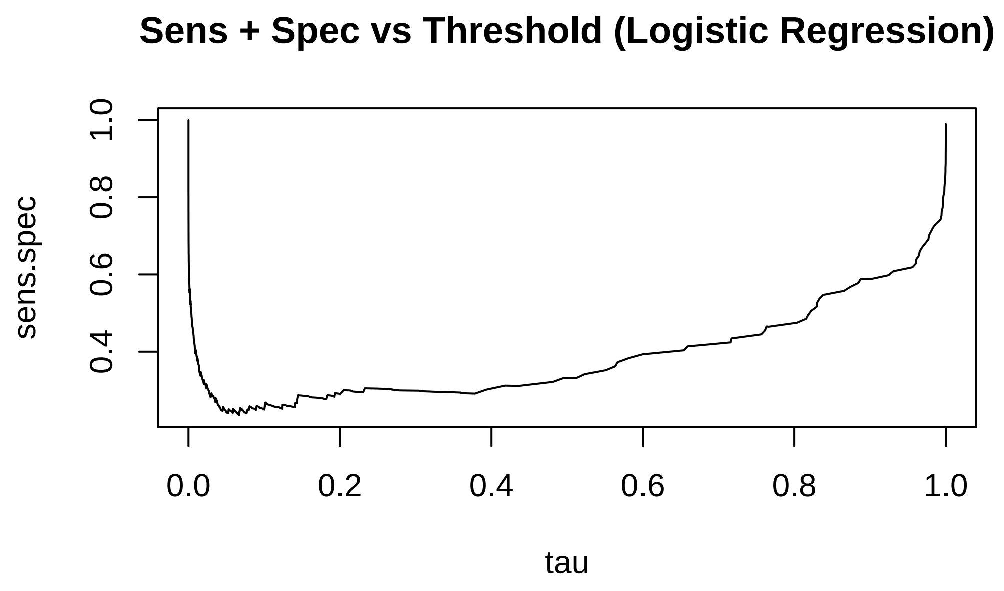
```r
library(ROCR)

# Retraining and fitting model on full training set
final.fit.log <- suppressWarnings(glm(y ~ ., data = data.train,
                                      family = "binomial"))
pred.log.obj <- prediction(fitted(final.fit.log), data.train$y)

# ROC plot
roc <- performance(pred.log.obj, "tpr", "fpr")
plot(roc, main = "ROC curve (Logistic Regression)")
```



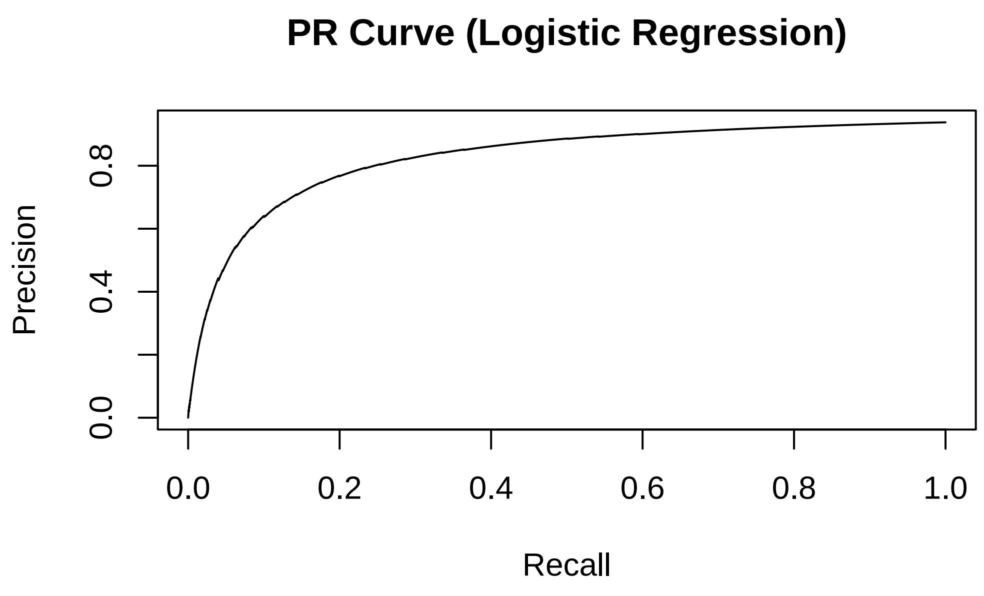
```
# Sens + spec vs tau splot
sens <- performance(pred.log.obj, "sens")
spec <- performance(pred.log.obj, "spec")
tau <- sens@x.values[[1]]
sens.spec <- sens@y.values[[1]] + spec@y.values[[1]]
plot(tau, sens.spec, type = "l",
     main = "Sens + Spec vs Threshold (Logistic Regression)")
points(tau[which.max(sens.spec)],
       sens.spec[which.max(sens.spec)], pch = 16, col = "blue")
```



```
# Optimal tau from sens + spec vs threshold plot
tau[which.max(sens.spec)]
```

```
Inf
```

```
# PR plot
pr <- performance(pred.log.obj, "prec", "rec")
plot(pr, main = "PR Curve (Logistic Regression)")
```

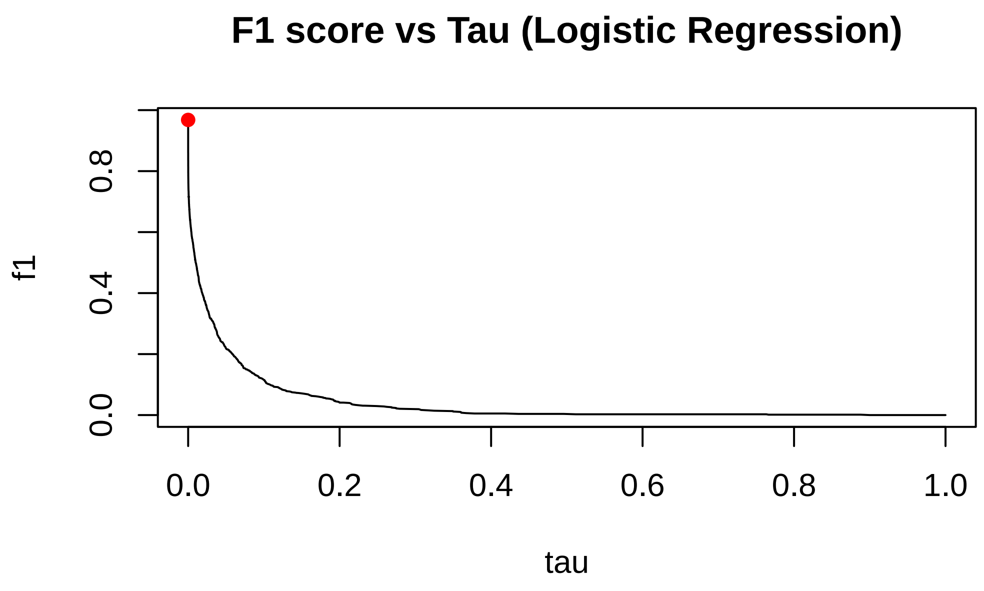


```
# Finding area under PR plot
aucpr <- performance(pred.log.obj, "aucpr")
aucpr@y.values
```

```
[[1]]
[1] 0.8260976
```

```
# F1 score vs threshold
f <- performance(pred.log.obj, "f")
tau <- f@x.values[[1]]
f1 <- f@y.values[[1]]

# Plotting F1 curve
best.pr <- which.max(f1)
plot(tau, f1, type = "l",
     main = "F1 score vs Tau (Logistic Regression)")
points(tau[best.pr], f1[best.pr], pch = 16, col = "red")
```



```
# Optimal tau from F1 vs threshold plot
tau[which.max(f1)]
```

```
        1951
2.220446e-16
```

```
pred.log <- ifelse(fitted(final.fit.log) > tau[best.pr], "break", "no_break")
pred.log <- factor(pred.log, levels = c("no_break","break"))
tab <- table(data.train$y, pred.log)
acc <- sum(diag(tab))/sum(tab)
sens <- tab[2,2]/sum(tab[2,])
spec <- tab[1,1]/sum(tab[1,])
perf.log.met.updated<-data.frame(Model = "Logistic Model (Updated Threshold)",
                                 `Tau.F1` = tau[best.pr],
                                 `Tau.Sens.Spec` = tau[which.max(sens.spec)],
                                 Accuracy = acc, Sensitivity = sens,
                                 Specificity = spec)
knitr::kable(perf.log.met.updated,
             caption =
               "Performance Metrics - Logistic Regression (Threshold Tuning)")
```

Table 2: Performance Metrics - Logistic Regression (Threshold Tuning)

	Model	Tau.F1	Tau.Sens.Spec	Accuracy	Sensitivity	Specificity
1951	Logistic Model (Updated Threshold)	0	Inf	0.0787956	1	0.0177596

- The ROC curve showed a low area under the curve and denotes weak diagnostic ability of the binary classifier as threshold τ is varied.
- As our model contains an imbalanced classification, `break`, the class of interest is rare.
- When sensitivity + specificity vs τ is plotted, we observe an optimal τ which maximizes sens + spec as `inf`. So this clearly shows that an ROC curve is not the right approach here. Hence a PR plot is implemented for our imbalanced dataset
- We obtain an area under the PR curve ≈ 0.82
- But our F1 vs Threshold plot gave an optimal $\tau \approx 0$ at the maximum F1 score.
- This shows that the data is highly imbalanced and finding optimal threshold using both Threshold tuning methods (Sens + Spec & F1) gave extreme values.
- Accuracy, Sensitivity and Specificity was calculated for the optimal τ from the F1 score vs Threshold plot and tabulated.

SVM

- We train an SVM models using all available features in the dataset using a Gaussian Radial Basis Function kernel (GRBF) with a 5-fold Cross validation and is repeated 10 times, tuning over a grid of selected set of C $\{1, 2, 5, 10, 20\}$ and σ $\{0.005, 0.010, 0.015, 0.020, 0.025, 0.030\}$
- In-sample predictive performance is evaluated by calculating *Accuracy*, *Sensitivity* and *Specificity* for each fold over the 10 replications. This ensures stable performance estimates across data variations.

```
library(kernlab)
# SVM model predictor and target data
x.svm <- data_processminer[,-1]
y.svm <- factor(data_processminer$y, levels = c("no_break", "break"))

# Setting seed for reproducibility
set.seed(24215155)

# test data
test.svm <- sample(1:n, n*0.2)
x.svm.test <- x.svm[test.svm,]
y.svm.test <- y.svm[test.svm]

# training and validation data
```

```r
train.svm <- setdiff(1:n, test.svm)
x.svm.train <- x.svm[train.svm, ]
y.svm.train <- y.svm[train.svm]
n.svm.train <- nrow(x.svm.train)

# Creating combination from C and sigma
C <- c(1, 2, 5, 10, 20)
sigma <- c(0.005, 0.010, 0.015, 0.020, 0.025, 0.030)
grid <- expand.grid(C, sigma)
colnames(grid) <- c("C", "sigma")

# size of grid
n.grid <- nrow(grid)

# Setting number of folds and replications
K <- 5
R <- 10

# Storing performance metrics
outSVM <- vector("list", R)

# For each replication
for(r in 1:R){
  # Storing performance metrics for each fold
  accSVM <- matrix(NA, K, n.grid)
  sensSVM <- matrix(NA, K, n.grid)
  specSVM <- matrix(NA, K, n.grid)
  # Randomizing fold data indexes to get random selection of data
  folds <- rep(1:K, ceiling(n.svm.train/K))
  folds <- sample(folds)
  folds <- folds[1:n.svm.train]
  # for each fold
  for (k in 1:K){
    # Setting indexes for training and validation data
    train.fold <- which(folds!=k)
    val.fold <- setdiff(1:n.svm.train, train.fold)

    # Standardizing training data and validation data
    x.train.fold.std <- scale(x.svm[train.fold,])
    x.val.fold.std <- scale(x.svm[val.fold,],
                            center = attr(x.train.fold.std, "scaled:center"),
                            scale = attr(x.train.fold.std, "scaled:scale"))
    # Fitting SVM model on training data
    for (j in 1:n.grid){
      fit.svm <- ksvm(x.train.fold.std, y.svm.train[train.fold], type = "C-svc",
                      kernel = "rbfdot",
                      C = grid$C[j], kpar = list(sigma = grid$sigma[j]))
      pred.svm <- predict(fit.svm, newdata = x.val.fold.std)
      tab <- table(y.svm.train[val.fold], pred.svm)
      sensSVM[k,j] <- tab[2,2]/sum(tab[2,])
      specSVM[k,j] <- tab[1,1]/sum(tab[1,])
      accSVM[k,j] <- sum(diag(tab))/sum(tab)
    }
  }
  # Store Accuracy, Sensitivity, Specificity
  outSVM[[r]] <- list(acc = accSVM, sens = sensSVM, spec = specSVM)
}

# mean accuracy, sensitivity and specificity across folds and replications
mean.SVM.acc <- rowMeans(sapply(outSVM, function(x) colMeans(x$acc)))
mean.SVM.sens <- rowMeans(sapply(outSVM, function(x) colMeans(x$sens)))
mean.SVM.spec <- rowMeans(sapply(outSVM, function(x) colMeans(x$spec)))

# Combine with grid
results.SVM <- cbind(grid, Accuracy = mean.SVM.acc,
                 Sensitivity = mean.SVM.sens,
                 Specificity = mean.SVM.spec)

# Best Accuracy Model
best.acc.SVM.model <- results.SVM[which.max(results.SVM$Accuracy),]
best.acc.SVM.model
```

```
  C sigma  Accuracy Sensitivity Specificity
1 1 0.005 0.9378618           0           1
```

```
# Best Sensitivity Model
best.sens.SVM.model <- results.SVM[which.max(results.SVM$Sensitivity),]
best.sens.SVM.model
```

```
    C sigma Accuracy Sensitivity Specificity
30 20  0.03  0.90884  0.03588262   0.9669762
```

```
# Best Specificity Model
best.spec.SVM.model <- results.SVM[which.max(results.SVM$Specificity),]
best.spec.SVM.model
```

```
  C sigma  Accuracy Sensitivity Specificity
1 1 0.005 0.9378618           0           1
```

```
# Best Model performance metrics
perf.svm.met <- data.frame(Model = "SVM (C=20, sigma=0.03)",
                           Accuracy = best.sens.SVM.model$Accuracy,
                           Sensitivity = best.sens.SVM.model$Sensitivity,
                           Specificity = best.sens.SVM.model$Specificity)

# Display performance metrics of best model
knitr::kable(perf.svm.met,
             caption = "Performance Metrics - GRBF SVM")
```

Table 3: Performance Metrics - GRBF SVM

Model	Accuracy	Sensitivity	Specificity
SVM (C=20, sigma=0.03)	0.90884	0.0358826	0.9669762

- Here we observe GRBF SVM model with $C = 1$ and $\sigma = 0.005$ having the best Accuracy (≈ 0.94) and Specificity (≈ 0.97).
- But, as our primary focus is on Sensitivity (Due to class imbalance), we consider the Model with $C = 20$ and $\sigma = 0.030$ that shows the best Sensitivity metric (≈ 0.035) without compromising much on Accuracy (≈ 0.91) and Specificity (≈ 0.97).
- Here, we can see that SVM model also suffered from the class imbalance issue, severely underdetecting `break` class. So we apply threshold tuning for the model.
- Training and Test Data is standardized on all the features using the training set standardizing scale. This is to prevent data leakage.
- Model is refitted on the full training data using `ksvm()` with the selected GRBF model parameters ($C = 20$ and $\sigma = 0.030$). Probabilities of class labels are returned rather than the class labels itself using argument `prob.model = TRUE`.

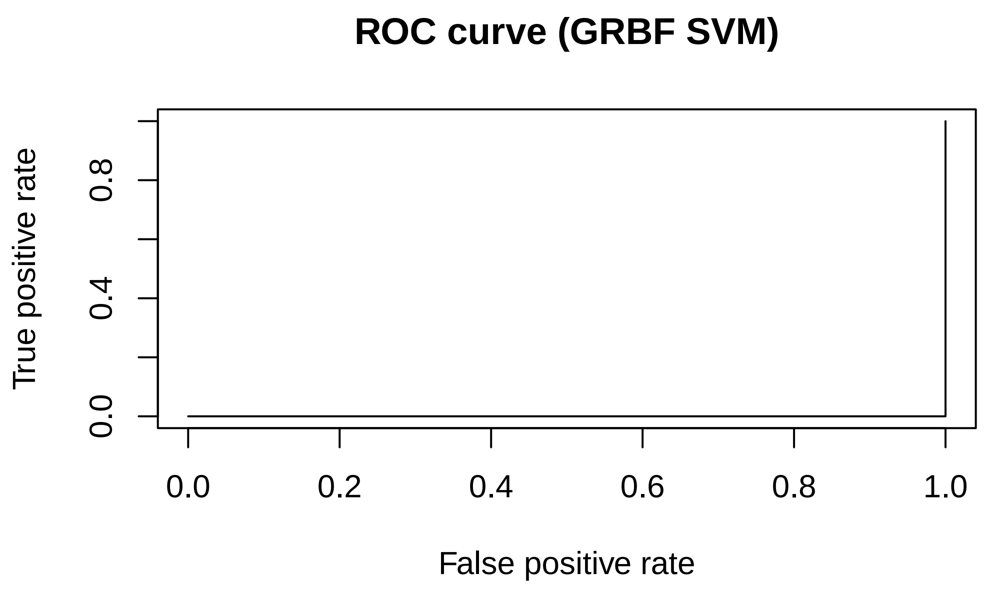
```
x.svm.scale <- scale(x.svm.train)
x.svm.train <- x.svm.scale

# Standardizing test data by using train data scaling
x.svm.test <- scale(x.svm.test,
                    center = attr(x.svm.scale, "scaled:center"),
                    scale = attr(x.svm.scale, "scaled:scale"))
```

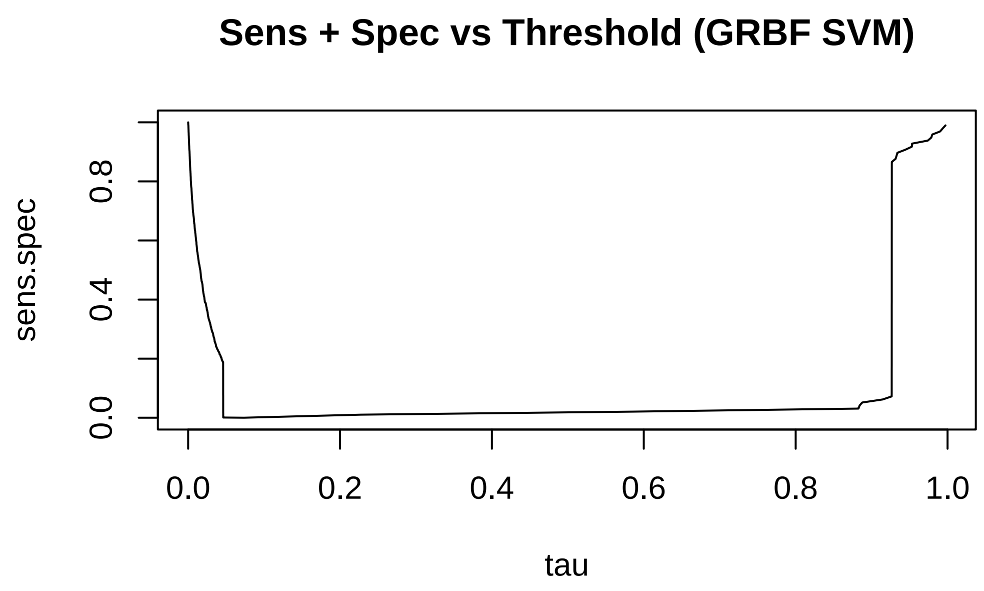
```r
# Retraining SVM Model on Train + Val data using selected SVM Model parameters
final.fit.svm <- ksvm (as.matrix(x.svm.train), y.svm.train, type = "C-svc",
                       kernel = "rbfdot",
                       C = best.sens.SVM.model$C,
                       kpar = list(sigma = best.sens.SVM.model$sigma),
                       # Enabling probability model when fitting SVM
                       prob.model = TRUE)
```

```r
# Refitting SVM model on full training set
pred.svm.prob <- predict(final.fit.svm, newdata = x.svm.train,
                         type = "probabilities")
pred.svm.obj <- prediction(pred.svm.prob[, "break"], y.svm.train)
roc <- performance(pred.svm.obj, "tpr", "fpr")

# ROC Plot
plot(roc, main = "ROC curve (GRBF SVM)")
```



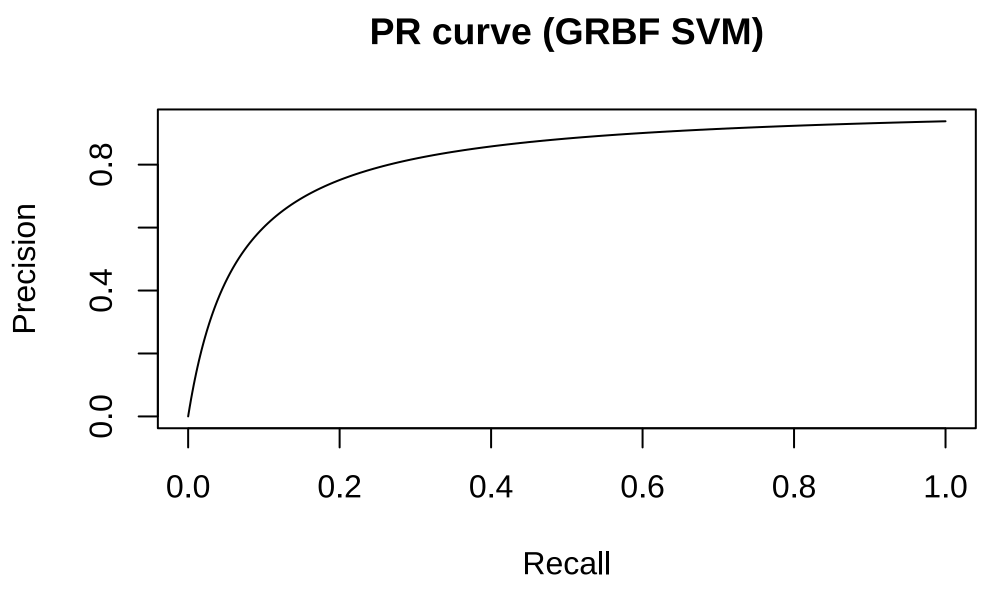
```r
# Sens + Spec vs tau plot
sens <- performance(pred.svm.obj, "sens")
spec <- performance(pred.svm.obj, "spec")
tau <- sens@x.values[[1]]
sens.spec <- sens@y.values[[1]] + spec@y.values[[1]]
plot(tau, sens.spec, type = "l",
     main = "Sens + Spec vs Threshold (GRBF SVM)")
points(tau[which.max(sens.spec)], sens.spec[which.max(sens.spec)],
       pch = 16, col = "blue")
```



```
# Optimal tau from sens + spec vs threshold plot
tau[which.max(sens.spec)]
```

```
[1] Inf
```

```
# PR plot
pr <- performance(pred.svm.obj, "prec", "rec")
plot(pr, , main = "PR curve (GRBF SVM)")
```

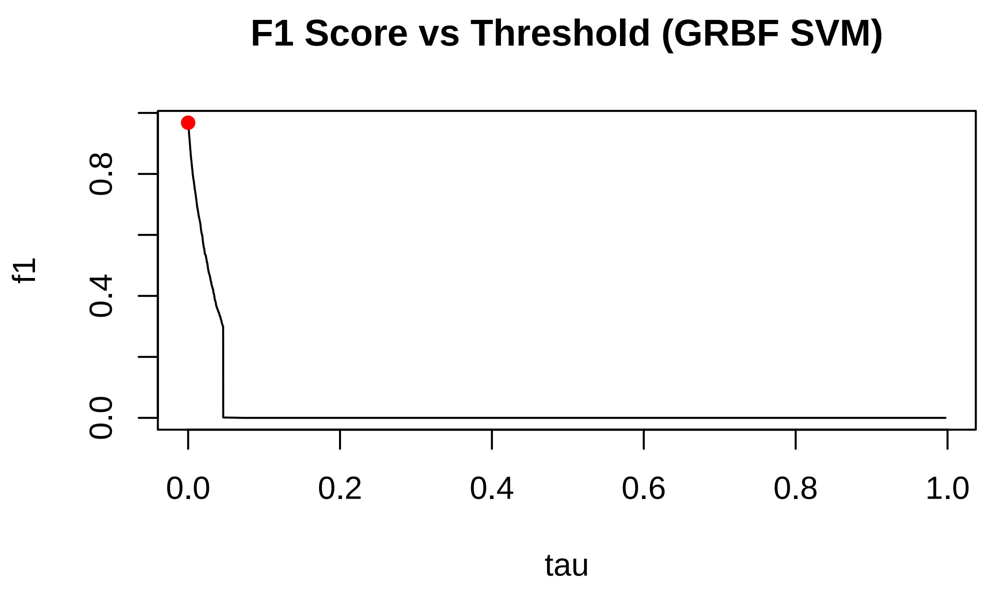


```
# Finding area under PR plot
aucpr <- performance(pred.svm.obj, "aucpr")
aucpr@y.values
```

```
[[1]]
[1] 0.8162549
```

```
# F1 score vs threshold
f <- performance(pred.svm.obj, "f")
tau <- f@x.values[[1]]
f1 <- f@y.values[[1]]
```

```r
# Plotting F1 score vs Threshold
best.pr <- which.max(f1)
plot(tau, f1, type = "l",
     main = "F1 Score vs Threshold (GRBF SVM)")
points(tau[best.pr], f1[best.pr], pch = 16, col = "red")
```



```r
# Optimal tau from F1 vs threshold plot
tau[which.max(f1)]
```

```
[1] 1.182957e-05
```

```r
# Calculating metrics for optimal tau
pred.svm <- ifelse(pred.svm.prob[, "break"] > tau[best.pr] ,
                   "break", "no_break")
pred.svm <- factor(pred.svm, levels = c("no_break","break"))
tab.svm <- table(y.svm.train, pred.svm)
acc.svm <- sum(diag(tab.svm))/sum(tab.svm)
sens.svm <- tab.svm[2,2]/sum(tab.svm[2,])
spec.svm <- tab.svm[1,1]/sum(tab.svm[1,])
perf.svm.met.updated <- data.frame(Model = "SVM (C=20, sigma=0.03)",
                                   `Tau.F1` = tau[best.pr],
                                   `Tau.Sens.Spec` = tau[which.max(sens.spec)],
                                   Accuracy = acc.svm, Sensitivity = sens.svm,
                                   Specificity = spec.svm)
# Displaying metric table for optimal tau
knitr::kable(perf.svm.met.updated,
             caption =
               "Performance Metrics - GRBF SVM (Threshold Tuning)")
```

Table 4: Performance Metrics - GRBF SVM (Threshold Tuning)

Model	Tau.F1	Tau.Sens.Spec	Accuracy	Sensitivity	Specificity
SVM (C=20, sigma=0.03)	1.18e-05	Inf	0.0627803	1	0.0006831

- The ROC curve showed almost zero area under the curve and denotes the very weak diagnostic ability of the binary classifier as threshold τ is varied.
- When sensitivity + specificity vs τ is plotted, we observe an optimal τ which maximizes sens + spec as `inf`. So this clearly shows that an ROC curve is not the right approach here (Similar to Logistic Model). Hence a PR plot is implemented for our imbalanced dataset.
- We obtain an area under the PR curve ≈ 0.81

- But our F1 vs Threshold plot gave an optimal $\tau \approx 1.2 \times 10^{-5}$ at the maximum F1 score.
- These results are almost identical to the one seen on the logistic regression model.
- Accuracy, Sensitivity and Specificity was calculated for the optimal τ from the F1 score vs Threshold plot and tabulated.
- We tabulate the performance metric results (Accuracy, Sensitivity & Specificity) we obtained before and after threshold tuning for both models (Logistic & SVM).

```r
# Before Model Tuning
knitr::kable(rbind(perf.log.met, perf.svm.met), digits = 4,
             caption = "Performance Metrics (Before F1 Tuning)")
```

Table 5: Performance Metrics (Before F1 Tuning)

Model	Accuracy	Sensitivity	Specificity
Logistic	0.9565	0.5068	0.9863
SVM (C=20, sigma=0.03)	0.9088	0.0359	0.9670

```r
# After Tuning
knitr::kable(rbind(perf.log.met.updated, perf.svm.met.updated),
             digits = 4,
             caption = "Performance Metrics (After F1 Tuning)")
```

Table 6: Performance Metrics (After F1 Tuning)

	Model	Tau.F1	Tau.Sens.Spec	Accuracy	Sensitivity	Specificity
1951	Logistic Model (Updated Threshold)	0	Inf	0.0788	1	0.0178
1	SVM (C=20, sigma=0.03)	0	Inf	0.0628	1	0.0007

- During implementation of threshold tuning for both models we observed the following issues:
  - When evaluating thresholds using ROC curves, maximizing sensitivity + specificity gave extreme τ values (`inf`), which causes predictions to only contain `no_break` classes - leading to 0 sensitivity
  - When evaluating thresholds using PR curves, maximizing F1 score gave very low extreme τ values (≈ 0) which causes all predictions to only contain `break` class - which leads to high sensitivity but very low specificity.
- Therefore, selecting a manual threshold based on our manufacturing needs could be one of the methods by which this issue could be rectified as:
  - Sensitivity is highly critical (breaks must not go unnoticed)
  - But excessive false postives (low specificity) could also lead to downtime
- To implement this, we calculate the performance metrics (*Accuracy, Sensitivity & Specificity*) for each of the model for threshold values ranging from 0 to 1 in 0.1 increments.

```r
# Define tau values to loop over
taus <- seq(0, 1, by = 0.1)

# Initialize empty data frames to store results
log_results <- data.frame()
svm_results <- data.frame()

# Loop over tau values and compute metrics for each
for (t in taus) {
  # Logistic Model Predictions
  pred.log <- ifelse(fitted(final.fit.log) > t, "break", "no_break")
  pred.log <- factor(pred.log, levels = c("no_break", "break"))
  tab.log <- table(data.train$y, pred.log)
  acc.log <- sum(diag(tab.log)) / sum(tab.log)
  sens.log <- tab.log[2,2] / sum(tab.log[2,])
  spec.log <- tab.log[1,1] / sum(tab.log[1,])
  log_results <- rbind(log_results, data.frame(
    Model = "Logistic",
    Tau = t,
    Accuracy = acc.log,
    Sensitivity = sens.log,
    Specificity = spec.log
  ))

  # SVM Predictions
  pred.svm <- ifelse(pred.svm.prob[, "break"] > t, "break", "no_break")
  pred.svm <- factor(pred.svm, levels = c("no_break", "break"))
  tab.svm <- table(y.svm.train, pred.svm)
  acc.svm <- sum(diag(tab.svm)) / sum(tab.svm)
  sens.svm <- tab.svm[2,2] / sum(tab.svm[2,])
  spec.svm <- tab.svm[1,1] / sum(tab.svm[1,])
  svm_results <- rbind(svm_results, data.frame(
    Model = "SVM (C=20, sigma=0.03)",
    Tau = t,
    Accuracy = acc.svm,
    Sensitivity = sens.svm,
    Specificity = spec.svm
  ))
}

# Display using knitr::kable
knitr::kable(log_results, digits = 4)
```

Model	Tau	Accuracy	Sensitivity	Specificity
Logistic	0.0	0.0621	1.0000	0.0000
Logistic	0.1	0.9283	0.8144	0.9358
Logistic	0.2	0.9622	0.7216	0.9781
Logistic	0.3	0.9725	0.7113	0.9898
Logistic	0.4	0.9782	0.6907	0.9973
Logistic	0.5	0.9782	0.6701	0.9986
Logistic	0.6	0.9737	0.5979	0.9986
Logistic	0.7	0.9725	0.5773	0.9986
Logistic	0.8	0.9699	0.5258	0.9993
Logistic	0.9	0.9635	0.4124	1.0000
Logistic	1.0	0.9379	0.0000	1.0000

```r
knitr::kable(svm_results, digits = 4)
```

Model	Tau	Accuracy	Sensitivity	Specificity
SVM (C=20, sigma=0.03)	0.0	0.0621	1.0000	0
SVM (C=20, sigma=0.03)	0.1	0.9994	0.9897	1
SVM (C=20, sigma=0.03)	0.2	0.9994	0.9897	1
SVM (C=20, sigma=0.03)	0.3	0.9987	0.9794	1
SVM (C=20, sigma=0.03)	0.4	0.9987	0.9794	1
SVM (C=20, sigma=0.03)	0.5	0.9987	0.9794	1
SVM (C=20, sigma=0.03)	0.6	0.9981	0.9691	1
SVM (C=20, sigma=0.03)	0.7	0.9981	0.9691	1
SVM (C=20, sigma=0.03)	0.8	0.9981	0.9691	1
SVM (C=20, sigma=0.03)	0.9	0.9962	0.9381	1
SVM (C=20, sigma=0.03)	1.0	0.9379	0.0000	1

- A manual threshold $\tau = 0.1$ was selected as a practical tradeoff as it provided a better balance between sensitivity and specificity in both models. At $\tau = 0.1$, high sensitivity values are shown across the two models without compromising on specificity.
- Comparing the two models at the selected manual threshold value, GRBF SVM model with $C = 20$ and $\sigma = 0.030$ showed the higher performance metrics ($Accuracy \approx 0.99$, $Sensitivity \approx 0.98$, $Specificity = 1$) on the training data when compared with the logistic regression model ($Accuracy \approx 0.93$, $Sensitivity \approx 0.8$, $Specificity \approx 0.93$).

- **So we select the SVM Model with Gaussian Radial Basis Function kernel (GRBF) having parameters $C = 20$ and $\sigma = 0.030$ as our best model. 0.1 is taken as the optimal threshold**.
- We apply the selected SVM model on the test set, with the manually chosen threshold $\tau = 0.1$ and computed the final performance metrics to evaluate generalized predictive performance.

```
chosen.tau = 0.1
# Getting predicted probabilities from final SVM model
pred.svm.prob <- predict(final.fit.svm, newdata = x.svm.test,
                          type = "probabilities")
# Classify based on threshold
pred.svm <- ifelse(pred.svm.prob[, "break"] > chosen.tau , "break", "no_break")
pred.svm <- factor(pred.svm, levels = c("no_break","break"))
# Confusion matrix
tab.svm <- table(y.svm.test, pred.svm)
# Performance
acc.svm <- sum(diag(tab.svm))/sum(tab.svm)
sens.svm <- tab.svm[2,2]/sum(tab.svm[2,])
spec.svm <- tab.svm[1,1]/sum(tab.svm[1,])
perf.svm.met.final <- data.frame(Model = "SVM Model(C=20, sigma=0.03)",
                                   `Tau` = chosen.tau,
                                   Accuracy = acc.svm, Sensitivity = sens.svm,
                                   Specificity = spec.svm)
knitr::kable(perf.svm.met.final, digits = 4,
             caption = "Predictive performance metrics (Test Data)")
```

Table 9: Predictive performance metrics (Test Data)

Model	Tau	Accuracy	Sensitivity	Specificity
SVM Model(C=20, sigma=0.03)	0.1	0.9051	0.7037	0.9201

```
tab.svm
```

```
          pred.svm
y.svm.test no_break break
  no_break      334    29
  break           8    19
```

Discussion - Reliability of the Selected SVM Model

- The predictive performance of the selected SVM model was evaluated on an unseen test dataset. Performance metrics and the confusion matrix are shown above.
- It achieved an **Accuracy of 90.5%, Sensitivity of 70.37% and Specificity of 92% with the selected threshold of $\tau = 0.2$**
- The results obtained indicate that the SVM model is reasonably accurate and most importantly, can identify a majority of `break` events in the manufacturing process (sensitivity).
- While this may not be a very high percentage, this trade-off minimizes undetected failures which if undetected can cause significant disruptions to the manufacturing process.
- Selecting the threshold manually was key in acheiving this trade-off, as optimal thresholds obtained from maximising F1 scores resulted in biases in performance metrics and unrealistic model behaviour, mainly due to the class imbalance.
- **Overall, the selected SVM model can be considered as reliable and well-calibrated for real world paper manufacturing `break` prediction.**