Predicting Paper Breaks in Industrial Process using SVM

Tuning C, sigma and tau to find best classifier

Jainel Liz Boban

Table of Contents

Load dataset 2

Train-Test splitting of data 2

Predictive performance metrics 2

Classifier training and tuning 3

Visualize predictive performance on validation set between classifiers 5

Boxplots of predictive performance metrics on validation 6

Plot of predictive performance based on tuning (C and σ) 7

Fitting best classifier = SVM (C = 50, σ=0.005) 9

Tuning tau for best selected model SVM (C = 50, σ=0.005) 10
- ROC curve 10
- PR curve 11
- Finding best tau for best model 11

Best classifier = SVM with C = 50 and σ=0.005 **13**

Load dataset

- Target variable y is extracted from column 'y' column of *data_processminer* data-set.
- We will extract input feature matrix x from the data-set loaded by removing first column (y).
- The target variable (y) is converted to factor to treat the problem as classification.
- Class "break" is chosen as positive class as it is more critical to detect "break" and False negatives (failing to predict a real break) are worse than false positives in this context.

```
# Load dataset
load("data_processminer.RData")
x <- data_processminer[,-1] # input features x1-x59
# Convert target variable to factor
data_processminer$y <- factor(data_processminer$y, levels = c("no_break", "break"))
y <- data_processminer$y # storing target separately
#check total number of observations in dataset
N <- nrow(data_processminer)
#Check class imbalance
table(y)
```

```
y
no_break    break
    1827      124
```

```
table(y)/N
```

```
y
  no_break      break
0.93644285 0.06355715
```

- It can be also seen from the from the data that there is a clear class imbalance with 93.6% of the data being "no_break" and only 6.4% as "break".

Train-Test splitting of data

- We will split the data so that 20% of observations are available for testing (i.e. unseen data on which we will assess test performance).
- Remaining 80% fill be used to train and tune the model.
- This portion of data will further be split into train and validation during cross-validation method.

```
#Set seed for reproducibility
set.seed(512)
# 80/20 split between test and train/val
test <-  sample(1:N, N*0.2)
data_test <-data_processminer[test,] # total test data
x_test <- x[test,] # test input
y_test <- y[test] # test target variable
train <- setdiff(1:N, test) # get remaining indices for train
data_train <-data_processminer[train,] #total train data
x_train <- x[train,] # train input
y_train <- y[train] # train target
N_train <- nrow(x_train) # get number of observations from train set
```

Predictive performance metrics

- To assess predictive performance of the classifiers, we get accuracy, sensitivity, specificity and f1 scores using a separate custom function *class_metrics* as shown in the code below.
- This function is then called during various such as cross-validation and testing to choose the best classifier and assess test performance respectively.

```
set.seed(512)
# function to compute metrics
class_metrics <- function(y, yhat){
  yhat <- factor(yhat, levels = c("no_break", "break"))
  tab <- table(y, yhat)
  acc <- sum(diag(tab)) / sum(tab) # accuracy
  sens <- tab["break", "break"] / sum(tab["break", ]) # sensitivity
  spec <- tab["no_break", "no_break"] / sum(tab["no_break",]) # specificity
  prec <- tab["break", "break"] / sum(tab[, "break"]) # precision
  f1 <- 2 * prec * sens / (prec + sens) # f1 score
  out <- c(acc = acc, sens = sens, spec = spec, f1 = f1)
return(out)
}
n_metrics <- 4 # number of metrics we get from 'out'
```

- The metrics are calculated as follows:

$$\text{Accuracy} = \frac{TP + TN}{\text{Total no: of samples}}$$

$$\text{Sensitivity/Recall/TPR} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TP}{TN + FP}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{F1 score} = \frac{2 * \text{Precision}}{\text{Precision+Recall}}$$

- where:
  - TP = True Positives (correctly classified as "break);
  - TN = True Negatives (correctly classified as "no_break");
  - FP = False Positives (other classes wrongly classified as "break");
  - FN = False Negatives (true "break" cases misclassified as "no_break).
- The metrics in this context are defined as follows:
  - **Accuracy:** The overall percentage of predictions that are correctly classified.
  - **Sensitivity:** Of those which are truly "break" cases, how many are correctly classified as positive
  - **Specificity:** Of those which are truly "no_break" cases, how many are correctly classified as negative.
  - **Precision:** Of those that are classified as "break", how many are truly "break".
- Since we are dealing with heavily imbalanced classification problem, accuracy is not an appropriate metric in choosing between classifiers. This is because the accuracy will always be high when most cases are predicted as "no_break" and can be misleading.
- Since the primary objective is to detect "break" which are both rare and critical, *sensitivity* becomes one of the most important metrics.
- High sensitivity is needed to ensure that we minimize false negatives and correctly catch true break cases, which is more critical than overall accuracy in this imbalanced problem."
- **F1 score** is the harmonic mean of precision and recall and balances between catching breaks (sensitivity/recall) and avoiding false breaks (precision).
- This balance is checked because if the model predicts too many false breaks, it may cause unnecessary alerts leading to operational inefficiency.
- However, we will be using it primarily for selecting optimal threshold (τ) later.

Classifier training and tuning

- For logistic regression, we will use all input features instead of a subset to account for all sensor readings. We use *glm()* to perform logistic regression.
- We will be using Gaussian Radial Basis Function kernel (GRBF) for the Support Vector Machine (SVM) because it is a flexible non-linear kernel that can model complex decision boundaries (the relationship between the input and target is likely non-linear). We use *kvsm()* from *kernlab* package for this.
- The kernal type is mentioned using argument *kernel = "rbfdot"* in the kvsm() function.
- GRBF Kernel is given by:

$$K(\mathbf{x}_i, \mathbf{x}_h) = \exp\left(-\sigma \|\mathbf{x}_i - \mathbf{x}_h\|^2\right)$$

- We will use K-fold validation to compare validation performance between classifiers (logistic and SVM) and tune hyperparameters C and σ for SVM.
- The parameter C is the cost parameter (penalty for misclassification) and we will tune this using a wide range of values as a higher cost than default is likely required since the model should not miss any breaks (a high C helps push the decision boundary harder to capture breaks).
- Similarly σ is the kernel width parameter, we might need smaller values than default as we need to detect rare minority class "break".
- We used a soft-margin SVM (*type = "C-svc"*) to allow for some misclassifications in the training data as in real-world scenarios like this problem is likely to be not perfectly separable especially when it is imbalanced.
- In this method, the data is divided into K=5 roughly equally sized folds.
- On each iteration one fold is dropped and the model is fit using remaining k-1 folds.
- The predictive performance is evaluated on dropped fold.
- For this the original training data is divided into train and test splits across each fold in each replicate.
- K-fold has been selected over leave-one-out method as it is computationally expense than k-fold and a simple hold-out method might lead to poor performance on test set as it learn patterns that do not generalize well.
- Use R replicates as shown in code helps reduce randomness in splitting and give more reliable model evaluation (else each fold might miss break cases due to imbalance).
- The best model is the one corresponding to the pair of hyperparameters and model type that gives the best predictive performance.
- Standardization is within each fold to avoid data leakage.
- We standardize the validation data inside each fold by using the statistics computed on the training data to avoid access to information about the validation data distribution before fitting.

```r
set.seed(512)

library(kernlab) # package to use svm

# values to check svm for various C and sigma by tuning
C <- c(1, 10, 20, 50, 100, 200, 500) # cost
sigma <- c(0.005, 0.010, 0.015, 0.020, 0.025, 0.030)

# create combinations of C and sigma for SVM tuning
grid <- expand.grid(C, sigma)
colnames(grid) <- c("C", "sigma")
n_mod <- nrow(grid) # number of combinations

R <- 10  # number of replicates
K <- 5   # number of folds
out <- vector("list", R)  # store metrics

for (r in 1:R) {
  metrics <- array(NA, dim = c(K ,n_mod+1 ,n_metrics))  # logistic + all SVM configurations
  folds <- rep(1:K, ceiling(N_train/K)) # create N_train+ fold numbers
  folds <- sample(folds) # random sampling
  folds <- folds[1:N_train] # make sure there are N_train data points
  for (k in 1:K) {
      train_fold <- which(folds != k) # training indices
      validation <- setdiff(1:N_train, train_fold) # test indices
      # standardize x1-x59
      x_train_fold_sc <- scale(x_train[train_fold, ])
      x_val_sc <- scale(x_train[validation, ], center = attr(x_train_fold_sc, "scaled:center"),
                  scale  = attr(x_train_fold_sc, "scaled:scale"))

      # create standardized data for glm: combine scaled X with y
      train_fold_sc <- data.frame(y = y_train[train_fold],x_train_fold_sc)
      val_fold_sc <- data.frame(y = y_train[validation],x_val_sc)
      # fit logistic classifier on the training data
      fit_log <- glm(y ~ ., data = train_fold_sc, family = "binomial", trace = FALSE)
      # predict on validation data in the dropped fold as probability
      pred_log <- predict(fit_log, type = "response", newdata = val_fold_sc)
      # convert to probability to labels by using  0.5 tau
      pred_log <- ifelse(pred_log > 0.5, "break", "no_break")
      metrics[k,1,] <- class_metrics(y = val_fold_sc$y, yhat = pred_log) # validation metrics

      # SVM tuning for C and sigma
      for ( j in 1:n_mod ) {
      # fit SVM for current pair of C and sigma
      fit <- ksvm(x_train_fold_sc,train_fold_sc$y, type = "C-svc",kernel = "rbfdot",
                C = grid$C[j], kpar = list(sigma = grid$sigma[j]))
          # predicted labels (0.5 tau by default)
          pred_svm <- predict(fit, newdata = x_val_sc)
          pred_svm <-factor(pred_svm, levels = levels(y_train))
          y_val <- factor(y_train[validation], levels = levels(y_train))
          metrics[k,j+1,] <- class_metrics(y = y_val, yhat = pred_svm) # validation metrics
      }
  }
  out[[r]] <- metrics
}
```

Visualize predictive performance on validation set between classifiers

- Since sensitivity is an important metric in this context, we will view various metrics at best sensitivity and also best f1 score to see how precision and sensitivity are balanced.

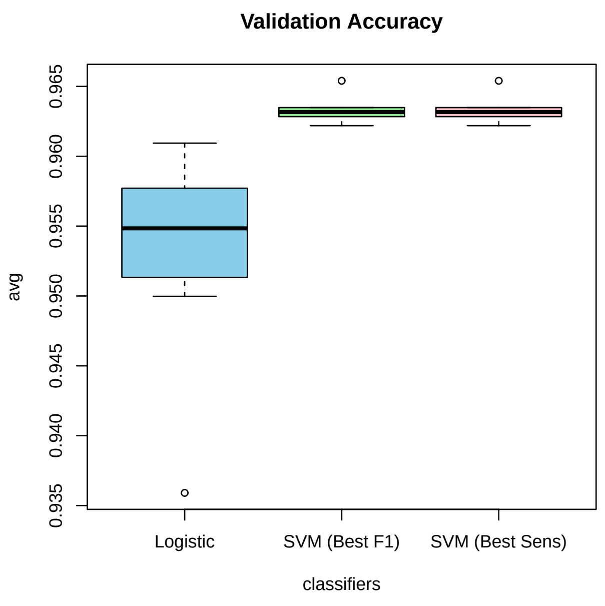
```
# combine metric from all repetitions
metrics_all <- array(NA, dim = c(K, n_mod + 1, n_metrics, R))
for (r in 1:R) {
  metrics_all[,,,r] <- out[[r]]
}
# take average by folds
avg <- apply(metrics_all, c(2, 3, 4), mean, na.rm = TRUE)
acc_mat <- t(avg[, 1, ])
sens_mat <- t(avg[, 2, ])
spec_mat <- t(avg[, 3, ])
f1_mat <- t(avg[, 4, ])

# find best model indices based on sensitivity and f1 score
best_sens <- which.max(colMeans(sens_mat[, -1])) + 1
best_f1 <- which.max(colMeans(f1_mat[, -1])) + 1

# ==== Mean metrics summary table ====
mean_metrics <- data.frame(Model = c("Logistic",paste0("SVM (Best Sens) [C=",
    grid$C[best_sens - 1], ",sigma=", grid$sigma[best_sens - 1], "]"),
    paste0("SVM (Best F1) [C=", grid$C[best_f1 - 1], ",sigma=", grid$sigma[best_f1 - 1], "]")),
  Accuracy = c(mean(acc_mat[, 1]), mean(acc_mat[, best_sens]), mean(acc_mat[, best_f1])),
  Sensitivity = c(mean(sens_mat[, 1]), mean(sens_mat[, best_sens]), mean(sens_mat[, best_f1])),
  Specificity = c(mean(spec_mat[, 1]), mean(spec_mat[, best_sens]), mean(spec_mat[, best_f1])),
  F1_Score = c(mean(f1_mat[, 1]), mean(f1_mat[, best_sens]), mean(f1_mat[, best_f1]))
)

# print metric results at best sensitivity and f1 scores
mean_metrics_rounded <- mean_metrics
mean_metrics_rounded[, -1] <- round(mean_metrics[, -1], 4)
print(mean_metrics_rounded)
```

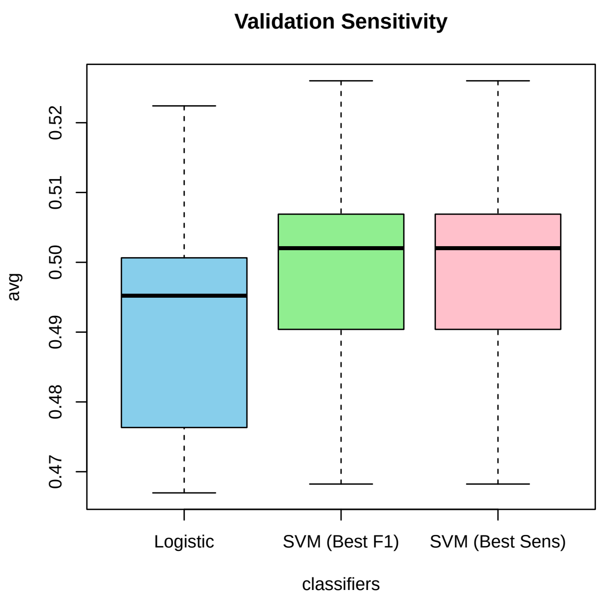
```
                              Model Accuracy Sensitivity Specificity F1_Score
1                          Logistic   0.9534      0.4914      0.9852   0.5752
```

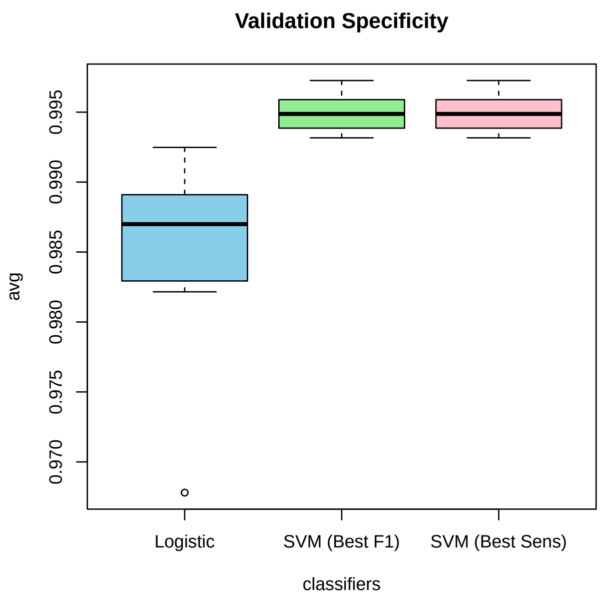
```
2 SVM (Best Sens) [C=50,sigma=0.005]   0.9633      0.5002      0.9949   0.6300
3   SVM (Best F1) [C=50,sigma=0.005]   0.9633      0.5002      0.9949   0.6300
```

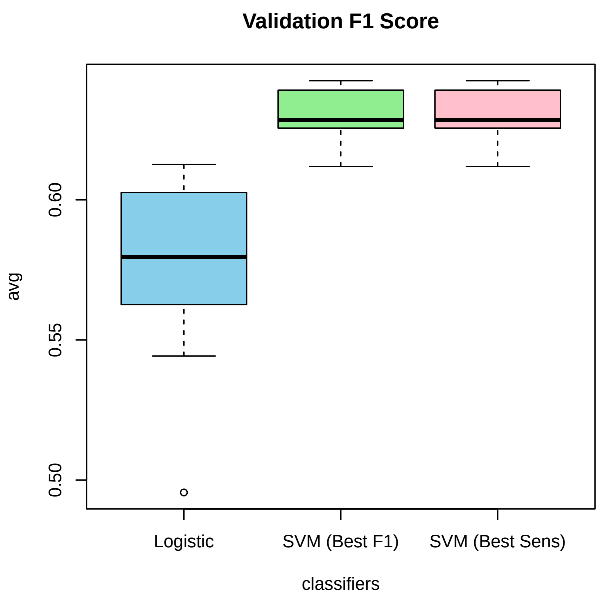
Boxplots of predictive performance metrics on validation

```r
# 3 classifiers
labels <- c("Logistic", "SVM (Best Sens)", "SVM (Best F1)")
# Prepare matrices for the 3 classifiers
acc_set <- cbind(acc_mat[, 1], acc_mat[, best_sens], acc_mat[, best_f1])
sens_set <- cbind(sens_mat[, 1], sens_mat[, best_sens], sens_mat[, best_f1])
spec_set <- cbind(spec_mat[, 1], spec_mat[, best_sens], spec_mat[, best_f1])
f1_set <- cbind(f1_mat[, 1], f1_mat[, best_sens], f1_mat[, best_f1])
mat_acc <- data.frame(avg = c(acc_set), classifiers = rep(labels, each = R))
mat_sens <- data.frame(avg = c(sens_set), classifiers = rep(labels, each = R))
mat_spec <- data.frame(avg = c(spec_set), classifiers = rep(labels, each = R))
mat_f1 <- data.frame(avg = c(f1_set), classifiers = rep(labels, each = R))
#boxplots
par(mfrow = c(2, 2))
boxplot(avg ~ classifiers, data = mat_acc, main = "Validation Accuracy",
        col = c("skyblue", "lightgreen", "pink"))
boxplot(avg ~ classifiers, data = mat_sens, main = "Validation Sensitivity",
        col = c("skyblue", "lightgreen", "pink"))
boxplot(avg ~ classifiers, data = mat_spec, main = "Validation Specificity",
        col = c("skyblue", "lightgreen", "pink"))
boxplot(avg ~ classifiers, data = mat_f1, main = "Validation F1 Score",
        col = c("skyblue", "lightgreen", "pink"))
```









Observations:

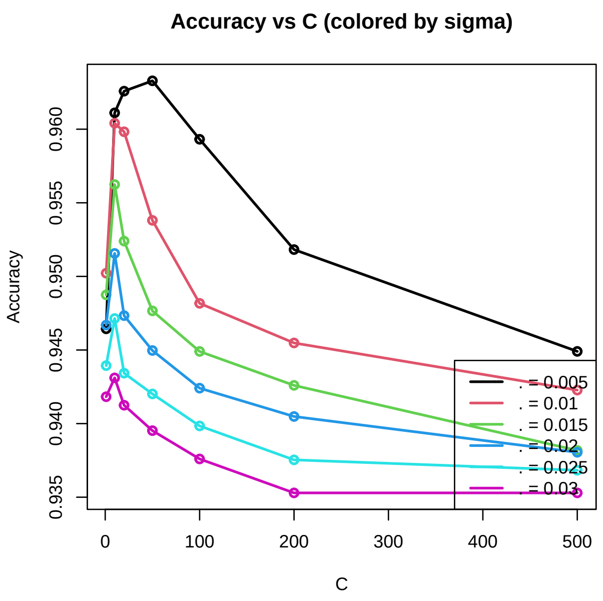
- From the table and boxplots we can choose **SVM (Best Sensitivity) with C = 50 and** σ=0.005 as the best model based on sensitivity. This will ensure detecting more breaks which is the most critical metric for our imbalanced dataset.
- It can be also seen that F1 is slightly lower than Best F1 model, but still better than Logistic.
- While Logistic has slightly higher specificity, the drop in sensitivity makes it less desirable in this context.
- Accuracy is high across all, but not very meaningful due to class imbalance. Hence, it has been given least focus for model selection.

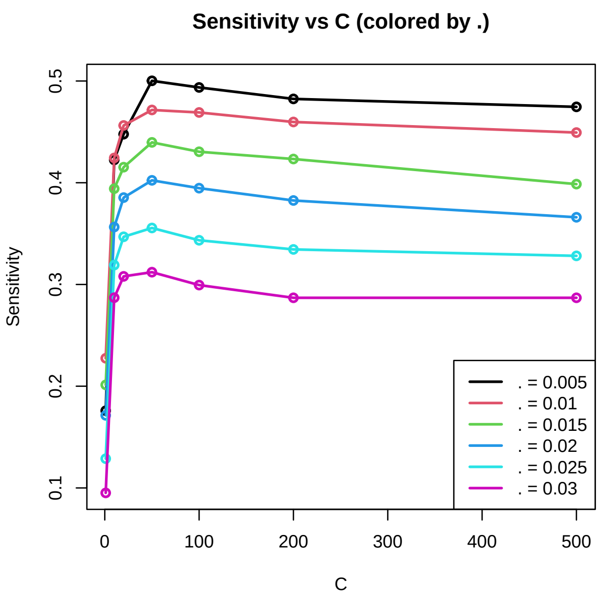
Plot of predictive performance based on tuning (C and σ)

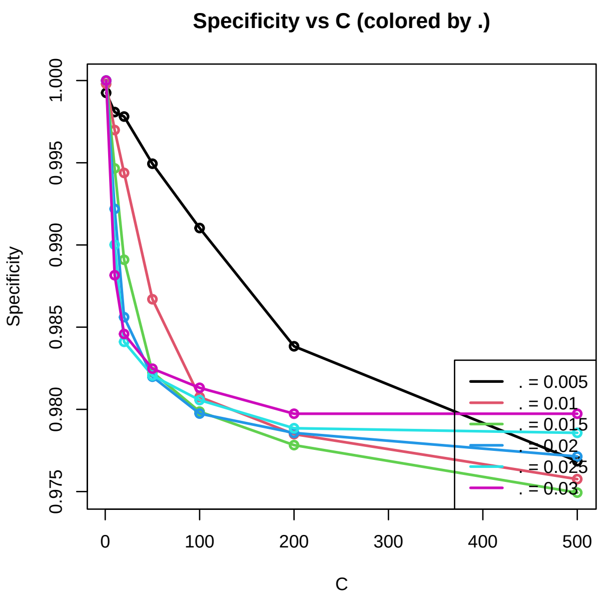
- The best model has high cost value (C=50) and low sigma value ($\sigma = 0.005$).
- This is as we had expected since using a large penalty C is required for the model to miss less breaks and small kernel width σ helps in detecting rare minority class "break".
- This can be further visualized using following plots of metrics vs C colored by σ.

```r
# average performance for each (C, sigma) pair
avg_acc <- colMeans(acc_mat[, -1], na.rm = TRUE)      # exclude logistic
avg_sens <- colMeans(sens_mat[, -1], na.rm = TRUE)
avg_spec <- colMeans(spec_mat[, -1], na.rm = TRUE)
```

```r
avg_f1 <- colMeans(f1_mat[, -1], na.rm = TRUE)
# add to single dataframes
acc_df <- data.frame(C = grid$C, sigma = grid$sigma, Accuracy = avg_acc)
sens_df <- data.frame(C = grid$C, sigma = grid$sigma, Sensitivity = avg_sens)
spec_df <- data.frame(C = grid$C, sigma = grid$sigma, Specificity = avg_spec)

# Plot metric vs C colored by sigma
par(mfrow = c(2, 2))
# Accuracy
plot(NULL, xlim = range(grid$C), ylim = range(acc_df$Accuracy), xlab = "C",
     ylab = "Accuracy", main = "Accuracy vs C (colored by sigma)")
for (s in unique(grid$sigma)) {
  lines(acc_df$C[acc_df$sigma == s], acc_df$Accuracy[acc_df$sigma == s],
        type = "o", col = which(unique(grid$sigma) == s), lwd = 2)}
legend("bottomright", legend = paste0("  = ", unique(grid$sigma)),
       col = 1:length(unique(grid$sigma)), lty = 1, lwd = 2)
# Sensitivity
plot(NULL, xlim = range(grid$C), ylim = range(sens_df$Sensitivity),
     xlab = "C", ylab = "Sensitivity", main = "Sensitivity vs C (colored by  )")
for (s in unique(grid$sigma)) {
  lines(sens_df$C[sens_df$sigma == s], sens_df$Sensitivity[sens_df$sigma == s],
        type = "o", col = which(unique(grid$sigma) == s), lwd = 2)}
legend("bottomright", legend = paste0("  = ", unique(grid$sigma)),
       col = 1:length(unique(grid$sigma)), lty = 1, lwd = 2)
# Specificity
plot(NULL, xlim = range(grid$C), ylim = range(spec_df$Specificity),
     xlab = "C", ylab = "Specificity", main = "Specificity vs C (colored by  )")
for (s in unique(grid$sigma)) {
  lines(spec_df$C[spec_df$sigma == s], spec_df$Specificity[spec_df$sigma == s],
        type = "o", col = which(unique(grid$sigma) == s), lwd = 2)}
legend("bottomright", legend = paste0("  = ", unique(grid$sigma)),
       col = 1:length(unique(grid$sigma)), lty = 1, lwd = 2)
```







Observations:

- It can be seen that smaller σ values (for example, $\sigma = 0.005$) consistently show higher sensitivity across most values of C, which is crucial for detecting rare break cases.
- As C increases, sensitivity initially improves and then declines slightly.
- Specificity decreases showing trade-off.
- The highest accuracy and specificity occur at lower C values, but this comes at the cost of reduced sensitivity. But since accuracy is almost high for all, this will not be an issue.
- $\sigma = 0.005$ with C around 50 offers a balance, achieving high sensitivity with reasonably good specificity and accuracy.
- This makes it ideal for our imbalanced classification problem.

Fitting best classifier = SVM (C = 50, σ=0.005)

- Focusing on high sensitivity, we select that SVM model with parameters C = 50 and σ=0.005.
- It was also seen that other metrics such as accuracy and F1 score of this combination on validation data gave sound results which makes it an apt hyperparameter pair.
- Here once again we standardize but on entire training set to prevent data leakage. This is then used to train the selected best classifier with best selected hypermarameter pair.
- The performance is enhanced by careful threshold tuning rather than relying on the default = 0.5. This

allows us to customize the trade-off between catching breaks and avoiding false positives based on our needs.

```
set.seed(512)

# standardize full training data before fitting best SVM
x_scale <- scale(data_train[, -1])
data_train[, -1] <- x_scale

# Fit SVM with best selected hyperparameters
best_svm <- ksvm(y ~ ., data = data_train, kernel = "rbfdot", kpar = list(sigma = 0.005),
                 C = 50, type = "C-svc", prob.model = TRUE)
```

Tuning tau for best selected model SVM (C = 50, σ=0.005)

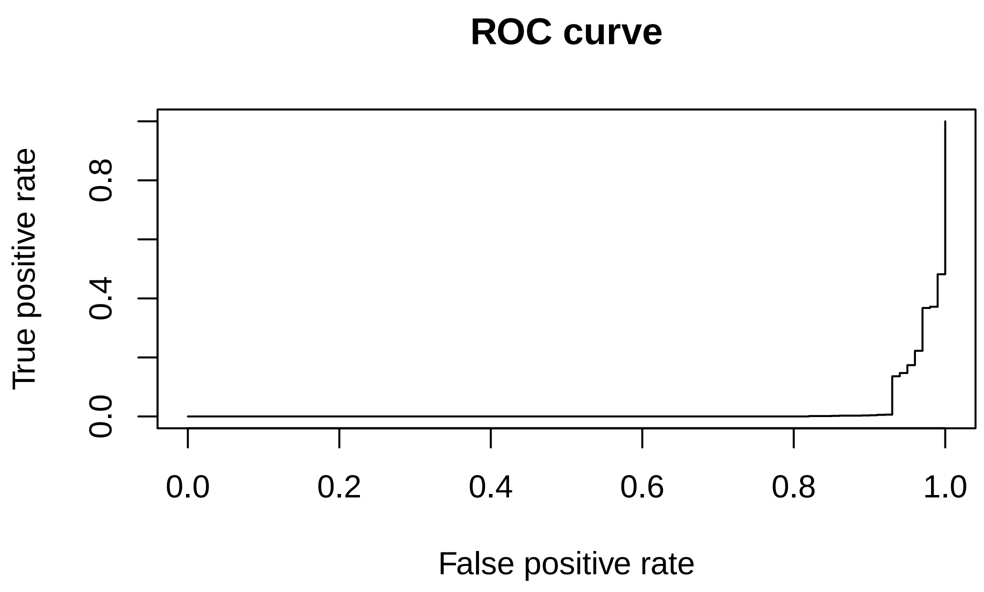
- Since our data set is imbalanced plotting ROC (Receiver operating characteristic) curve will not be a sensible choice as it assumes that the two classes are balanced. This leads to ROC curve as shown.

```
library(ROCR) # ROC and PR etc

# predict to get probabilities on training data itself
pred_train <- predict(best_svm, newdata = data_train, type = "probabilities")
pred_break_train <- pred_train[, "break"]
# create prediction object for ROC and PR
pred_obj <- prediction(pred_break_train, data_train$y)
```

ROC curve

The ROC curve is shown as a function of τ.

```
# ROC curve
roc <- performance(pred_obj, "tpr", "fpr")
plot(roc, main = "ROC curve")
```



```
# compute the area under the ROC curve
auc <- performance(pred_obj, "auc")
auc@y.values
```

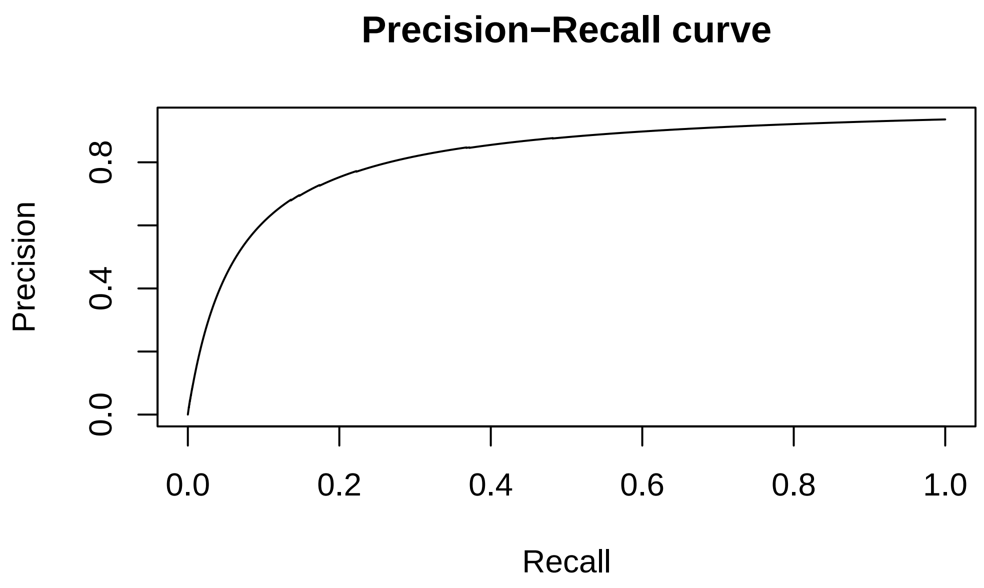
```
[[1]]
[1] 0.01934292
```

- It can be seen that area under ROC curve is very low and by ROC classifier performs poorly at most thresholds, with true positive rate being low until very high false positive rates.
- This is expected due to class imbalance and it would be practically more useful to use PR curve than ROC at evaluating performance on imbalanced datasets.

PR curve

- It would be more apt to view PR (Precision-Recall) as it is more focused on the positive class, and do not account for true negatives.

```
pr <- performance(pred_obj, "prec", "rec")
plot(pr, main = "Precision-Recall curve")
```



```
# area under pr curve
aucpr <- performance(pred_obj, "aucpr")
aucpr@y.values[[1]]
```

```
[1] 0.8160332
```

- The PR curve is consistently high for this selected classifier showing a strong trade-off between precision and recall with a good curve shape even in the presence of class imbalance.
- This confirms the model is effective at identifying the minority class ("break").
- Area under PR curve (approx. 0.82) is also high (which is desirable).
- A good threshold τ can significantly improve predictive performance metrics and this PR curve proves that it is worth tuning τ carefully.

Finding best tau for best model

```
# F1 score
perf_f <- performance(pred_obj, "f")
tau_f <- perf_f@x.values[[1]]
f1_vals <- perf_f@y.values[[1]]
# get tau at best f1_score
best_tau_f1 <- tau_f[which.max(f1_vals)]
# sensitivity+specificity
sens <- performance(pred_obj, "sens")
spec <- performance(pred_obj, "spec")
tau_roc <- sens@x.values[[1]]
sens_vals <- sens@y.values[[1]]
spec_vals <- spec@y.values[[1]]
```

```r
sum_sens_spec <- sens_vals + spec_vals
# get tau at best sensitivity+specificity
best_tau_sensspec <- tau_roc[which.max(sum_sens_spec)]
# confusion matrix at best_tau_f1
pred_f1 <- factor(ifelse(pred_break_train > best_tau_f1, "break", "no_break"),
                  levels = c("no_break", "break"))
cat("Confusion matrix (best F1 tau = ", round(best_tau_f1, 3), "):\n")
```

```
Confusion matrix (best F1 tau =  0 ):
```

```r
print(table(Actual = data_train$y, Predicted = pred_f1))
```

```
          Predicted
Actual     no_break break
  no_break        1  1460
  break           0   100
```

```r
# confusion matrix at best_tau_sensspec
pred_sensspec <- factor(ifelse(pred_break_train > best_tau_sensspec, "break", "no_break"),
                        levels = c("no_break", "break"))
cat("Confusion matrix (best Sens+Spec tau = ", round(best_tau_sensspec, 3), "):\n")
```

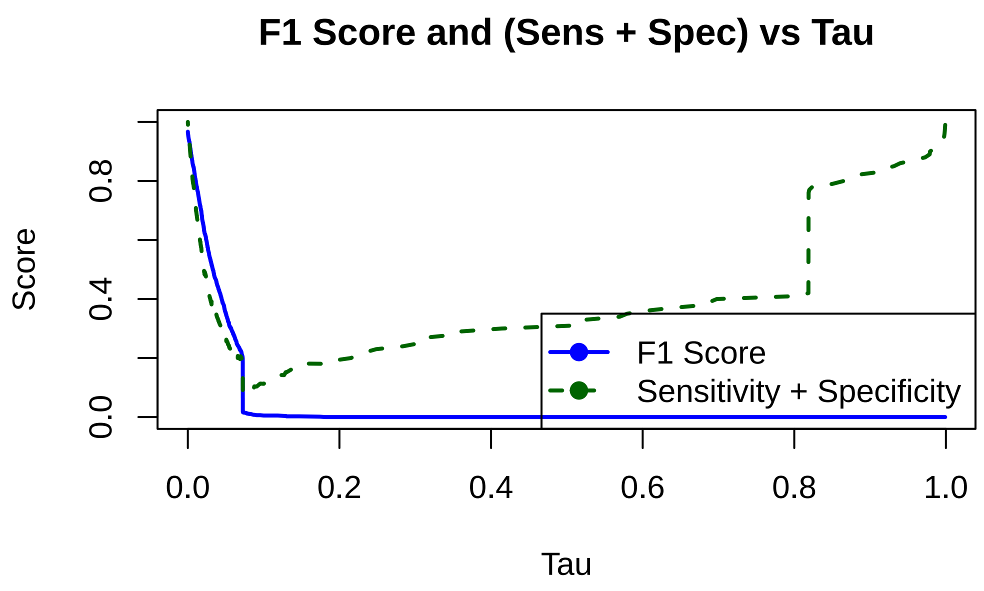
```
Confusion matrix (best Sens+Spec tau =  Inf ):
```

```r
print(table(Actual = data_train$y, Predicted = pred_sensspec))
```

```
          Predicted
Actual     no_break break
  no_break     1461     0
  break         100     0
```

- From the above confusion matrices, it can be seen that using usual metrics such as best τ at best f1 score and (sensitivity+specificity) are at extreme values (0 and inf) and is not performing well at all.
- Hence, we will manually choose a τ using the plots of F1 and (Sensitivity+Specificity) vs τ as shown.

```r
# Plot F1 and Sens+Spec vs Tau
plot(tau_f, f1_vals, type = "l", col = "blue", lwd = 2,xlab = "Tau", ylab = "Score",
     ylim = c(0, 1), main = "F1 Score and (Sens + Spec) vs Tau")
points(best_tau_f1, max(f1_vals), pch = 19, col = "red") # Add best threshold points
points(best_tau_sensspec, max(sum_sens_spec), pch = 19, col = "orange")
lines(tau_roc, sum_sens_spec, col = "darkgreen", lwd = 2, lty = 2)
legend("bottomright",legend = c("F1 Score", "Sensitivity + Specificity"),
       col = c("blue", "darkgreen"),lty = c(1, 2), lwd = 2, pch = 19)
```



It can be seen that for very small values of τ such as 0.075, both F1 score and (sensitivity+specificity) is good. Confusion matrix and predictive performance on full train set using this new tau can be seen:

```
#confusion matrix at manual threshold  = 0.075
pred_train_new <- factor(ifelse(pred_break_train > 0.075, "break", "no_break"),
                         levels = c("no_break", "break"))
cat("Confusion matrix (0.075 ):\n")
```

```
Confusion matrix (0.075 ):
```

```
print(table(Actual = data_train$y, Predicted = pred_train_new))
```

```
          Predicted
Actual     no_break break
  no_break     1450    11
  break           7    93
```

```
class_metrics(data_train$y,pred_train_new)[1:3]
```

```
      acc      sens      spec
0.9884689 0.9300000 0.9924709
```

Observations:

It can be seen that the SVM model with $\tau = 0.075$ performs very well on the training data showing a strong balance between sensitivity and specificity, making it highly suitable for predicting paper breaks in our classification problem.

Best classifier = SVM with C = 50 and σ=0.005

- We standardize the test data by using the statistics computed on the training data and prevent data leakage by avoiding access to information about the test data distribution.

```
#standardize test data separately
data_test[,-1] <- scale(data_test[,-1], center = attr(x_scale, "scaled:center"),
                        scale = attr(x_scale, "scaled:scale"))

# Using final model to predict probabilities on test data
pred_test <- predict(best_svm, newdata = data_test, type = "probabilities")
# get probability for the positive class = "break"
pred_break_test <- pred_test[, "break"]
# Use selected tau =0.075 to predict labels on test data
```

```r
selected_tau <- 0.075
pred_test <- factor(ifelse(pred_break_test > selected_tau, "break", "no_break"),
                    levels = c("no_break", "break"))

# Performance evaluation
table(Actual = data_test$y, Predicted = pred_test) # confusion matrix
```

```
          Predicted
Actual     no_break break
  no_break      333    33
  break           6    18
```

```r
class_metrics(data_test$y, pred_test)[1:3] # predictive performance metrics
```

```
      acc      sens      spec 
0.9000000 0.7500000 0.9098361 
```

Observations on predictive performance:

- In real-world scenario, breaks are rare and dataset will be imbalanced like our dataset and the selected model shows good potential for predicting paper breaks.
- A sensitivity of 75% shows that the model is able to correctly identify majority of true break cases. This is critical as missing a break could lead to higher operational costs, material waste and machinery damage.
- 91% specificity shows the model's ability to correctly recognize no-break cases which minimize false alarms. This is important as too many false positives could lead to unnecessary interruptions and hence again higher costs.
- Accuracy of 90% is good shows the model's reliability across both "break" and "no_break" cases.
- Repeated cross-validation and testing show consistent results increasing confidence in the model's generalizability to unseen data.
- Overall, the model works well and can be trusted to predict paper breaks in real-world situations. It correctly catches most actual breaks (good sensitivity) while avoiding too many false alarms (good specificity).