STAT 542 / CS 598: Homework 7

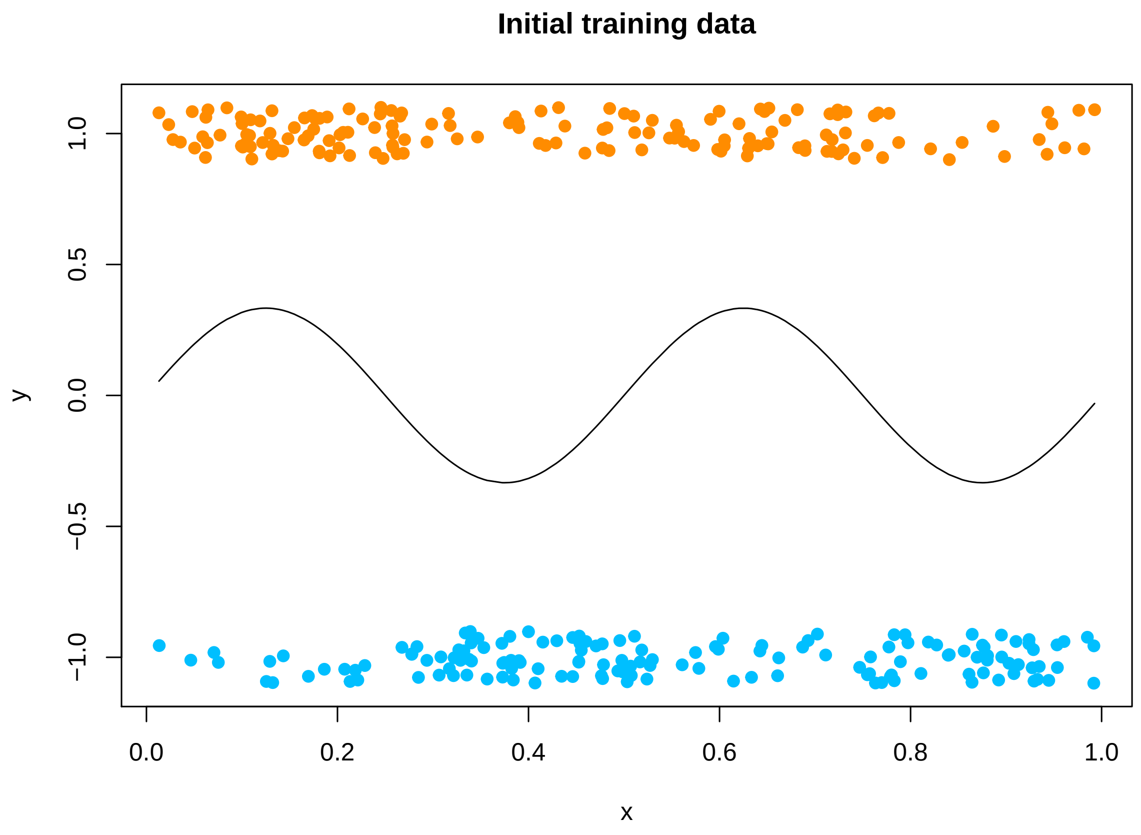
Pushpit Saxena (netid: pushpit2)

Contents

Question 1: AdaBoost with stump model

- Data Generation:

```r
set.seed(1)
n = 300
x = runif(n)
py <- function(x) sin(4*pi*x)/3 + 0.5
y = (rbinom(n, 1, py(x))-0.5)*2
z<-y + 0.1*runif(n, -1, 1)
plot(x, y + 0.1*runif(n, -1, 1), ylim = c(-1.1, 1.1), pch = 19,
     col = ifelse(y == 1, "darkorange", "deepskyblue"), ylab = "y",
     main="Initial training data")
lines(sort(x), py(x)[order(x)] - 0.5)
```



```r
  # dd <- rbinom(n, 1, py(x))
  # cbind(t(py(x)), t(dd))
  testx = seq(0, 1, length.out = 1000)
  testy = (rbinom(1000, 1, py(testx))-0.5)*2
```

Write a function to fit the stump model with subject weights:

- **Step 1**: Search for a splitting rule $\mathbf{1}(x \le c)$ that will maximize the weighted reduction of Gini impurity.
- **Step 2**: Calculate the left and the right node predictions $f_L, f_R \in \{-1, 1\}$ respectively.

Answer:

- There are following functions:
  - **calc.gini** : To calculate the gini score for a given cutoff (c). It also calculate the predictions for left and right child node also (for future predictions if this cut-off point is selected)
  - **myadaBoost**: Implemented the Adaboost algo based on the weak learners(stump model with cutoff identified by selecting the value of x which is maximizing the gini score).
  - **myadaBoost.predict** : Predict function to perform prediction on test data using the trained adaboost model.
  - I have packaged the **x**, **y**, and **w(weights)** in a data frame for easier processing

```r
calc.gini <- function(i, data) {
    c <- data[i, 1]
    L <- data[data[, 1] <= c,]
    R <- data[data[, 1] > c,]

    p_hat_L <- sum(L[L[, 2] == 1, 3])/sum(L[,3])
    p_hat_R <- sum(R[R[, 2] == 1, 3])/sum(R[,3])

    gini_L = p_hat_L * (1 - p_hat_L)
    gini_R = p_hat_R * (1 - p_hat_R)
    score = -(sum(L[,3]) * gini_L) - (sum(R[,3]) * gini_R)
    return(cbind(score, ifelse(p_hat_L >= 0.5, 1, -1),
                 ifelse(p_hat_R >= 0.5, 1, -1)))
}

myadaBoost <- function(x, y,num_trees = 500, delta=0.8) {
  fx <- matrix(0, nrow = n, ncol=num_trees)
  stumpModels <- matrix(0, nrow=num_trees, ncol = 3)
  alphas <- rep(0, num_trees)
  n <- length(x)
  w <- rep(1/n, n)
  data <- data.frame(x=x,y=y,w=w)
  expo_loss <- rep(0, num_trees)
  cumm_pred <- rep(0, num_trees)
  for (i in 1:num_trees) {
    # Fitting a weak learner stump model
    giniAll <- t(as.matrix(sapply(1:n, function(i) return (calc.gini(i, data)))))
    bestGini <- giniAll[which.max(giniAll[,1]),]
    cutOff <- data[which.max(giniAll[,1]),1]
    stumpModels[i,] <- cbind(cutOff, bestGini[2], bestGini[3])

    # Prediction for stump model in this iteration
    ypred <- ifelse(data[,1] > cutOff, bestGini[3], bestGini[2])
    # Eta
    eta <- sum(data[which(y != ypred),3])

    # Alpha
    alpha <-  0.5 * log((1-eta)/eta)
    alphas[i] <- alpha

    # Weight updation. Also used shrinkage(delta)
    w_1 <- data[,3] * exp(-alpha*delta * y * ypred)
    # Normalizing weights
    w_1 <- w_1/sum(w_1)
    fx[,i] <- delta*alpha*ypred
    expo_loss[i] <- mean(exp(-y * rowSums(fx)))
    data[,3] <- w_1
  }

  # Final model
  Fx <- rowSums(fx)
  return(list("Fx" = Fx, "fitted" = sign(Fx),
              "GX" = stumpModels, "alphas"=alphas,
              "expo-loss" = expo_loss))
}

myadaBoost.predict <- function(model, xtest, ytest,delta) {
  n <- length(xtest)
  ypred <- sapply(1:n, function(i) {
              pred <- sum(model$alphas * delta *
                    (ifelse (xtest[i] > model$GX[,1],
                   model$GX[,3], model$GX[,2])))
            return (pred)
          })
  return (ypred)
}
```

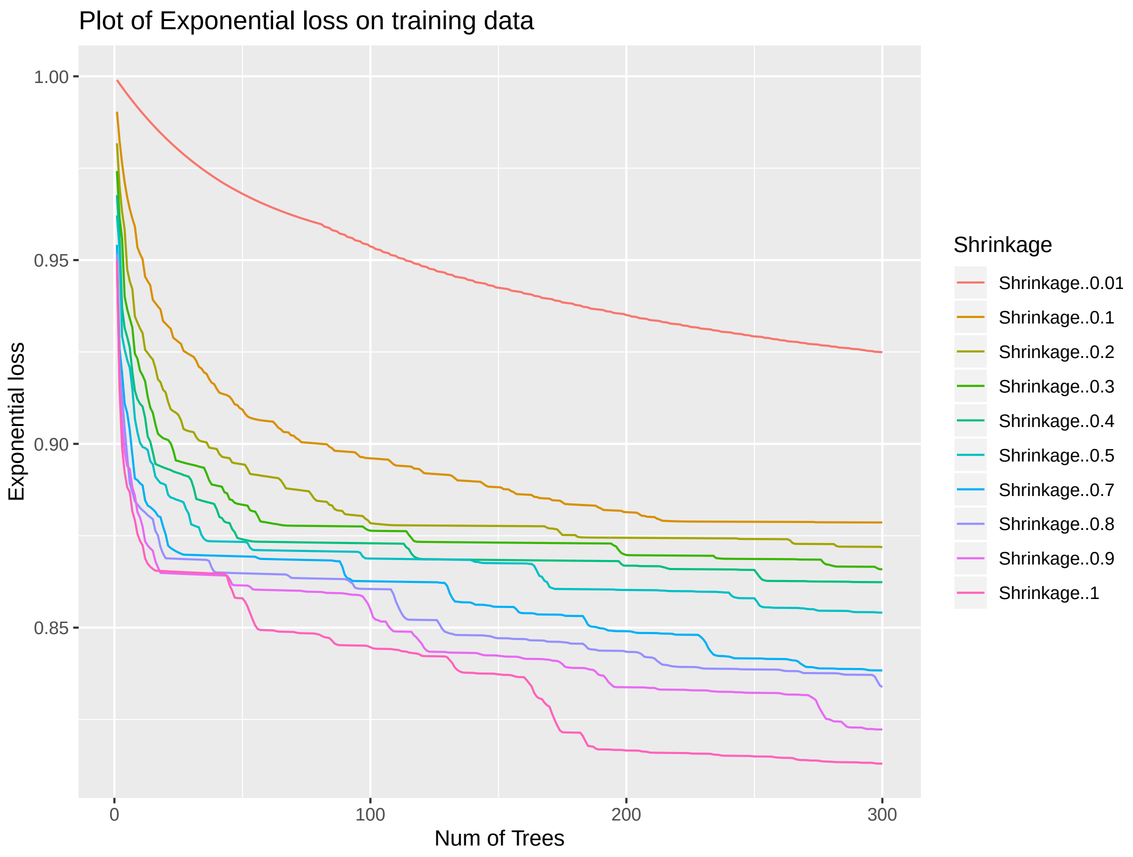
Based on the AdaBoost algorithm, write your own code to fit the classification model

- Implement a **shrinkage** factor δ, which is commonly used in boosting algorithms.
  - I have added a **shrinkage** factor (see **delta** variable which is passed to adaboost function)

```
num_trees <- 300
delta <- c(0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1)
models.expo_loss <- data.frame("Num_Trees" = c(1:num_trees))
models <- rep(NA, length(delta))
for (i in 1:length(delta)) {
  adaModel <- myadaBoost(x, y, num_trees=num_trees, delta=delta[i])
  models[i] <- list(adaModel)
  models.expo_loss[, paste("Shrinkage ", delta[i])] <- adaModel$`expo-loss`
}
```

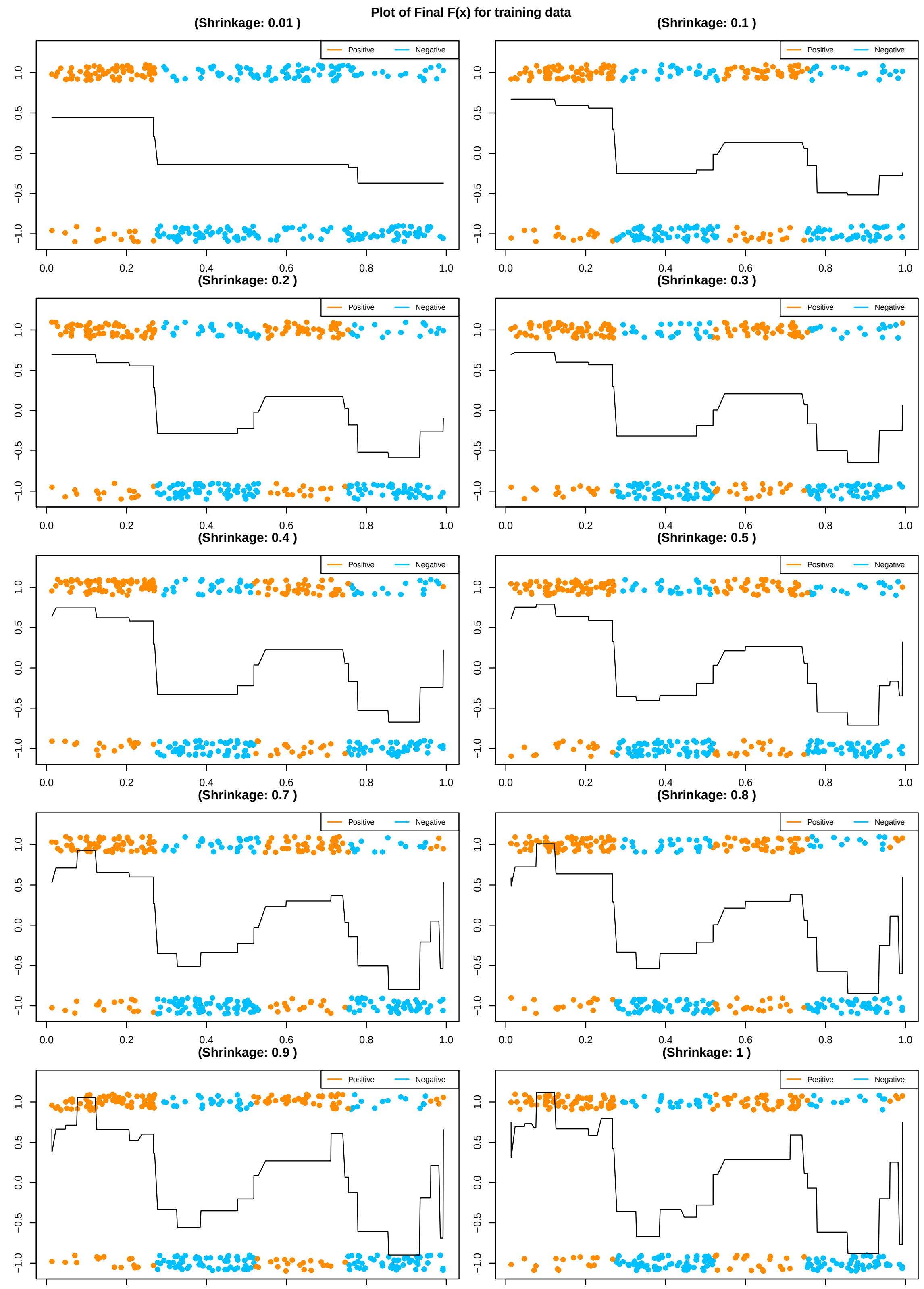
- Plot the exponential loss $n^{-1}\sum_{i=1}\exp\{-y_i\delta\sum_k \alpha_k f_k(x_i)\}$ over the number of trees and comment on your findings.
  - Below is the plot for exponential loss (calculated as per the formula given) over the number of trees while training the model.

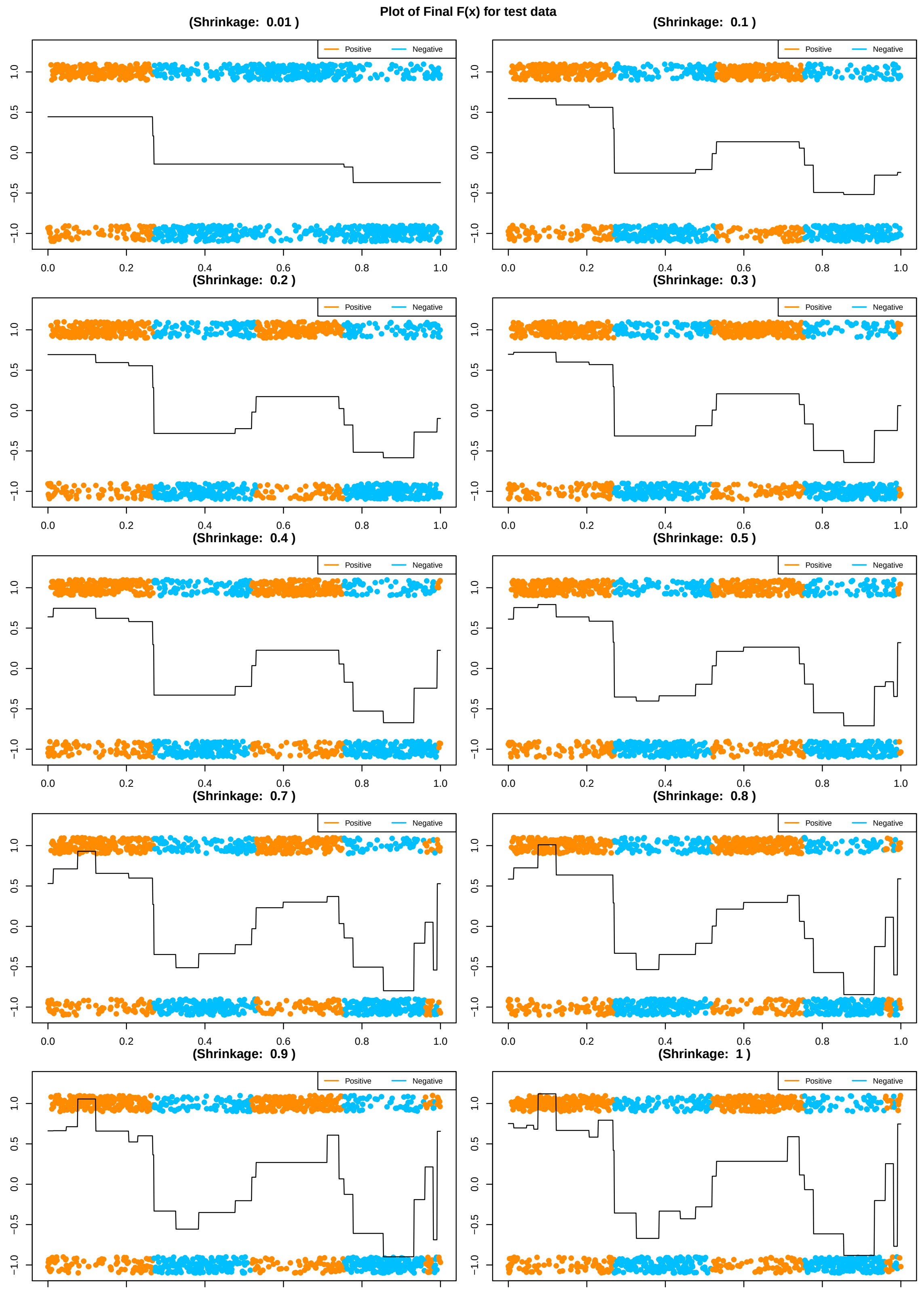
– As it is evident the exponential loss is going down as the number of trees increases in the model, which can be reasoned as having more number of decision stumps in our adaboost classification model leads to lower generalization error and better fit. I have done experiment with setting `num_trees=5000` (in above code; experiment itself and its corresponding plots are not included in the report, but can be easily re-generated using the code at the bottom of RMD code file) and the exponential loss was still decreasing and accuracy went in higher 90% (which is kind of an indication of overfitting). So to conclude it is possible that if we keep on increasing the number of decision stumps the loss will keep on decreasing and leads model to overfit training data. The number of trees leads to overfitting depends on the **Shrinkage** factor (discussed later). I have noticed with high value for shrinkage the overfitting happens relatively quickly with lesser number of trees. One other thing I noticed that with smaller Shrinkage values, if we keep the number of trees constant, we may reach a point of diminishing gains even with relatively smaller number of trees (indicative of underfit). So both `num_trees` and `Shrinkage` hyper parameters here are dependent on each other. For this report I have choosen `num_trees=300` (based on the performance and training time trade-off)



- Try a few different `shrinkage` factors and comment on your findings.
  - I have tried 10 different values for Shrinkage (defined as **delta** vector in the code)
  - The basic idea behind using a `shrinkage` factor is to slow down learning rate which in turns helps to reduce overfitting. Adaboost with weak predictors (decision stumps) manages to reach lower generalization error by not overfitting early with the use of Shrinkage factor, which is not the case with vanilla decision trees.
  - As can be evident with exponential-loss plot above and Final F(x) function plots (later), it can be seen that higher values of Shrinkage leads to better fit of the training data. But as we go higher and higher for Shrinkage value, it can leads to overfitting, as from the plots (exponential-loss plot and F(x) plot), for the $\delta = 1$, we can clearly see the model is overfitting the data (It is also evident by seeing the exponential-loss plot (see later) for test data, where for $\delta = 1$, the loss actually went up).

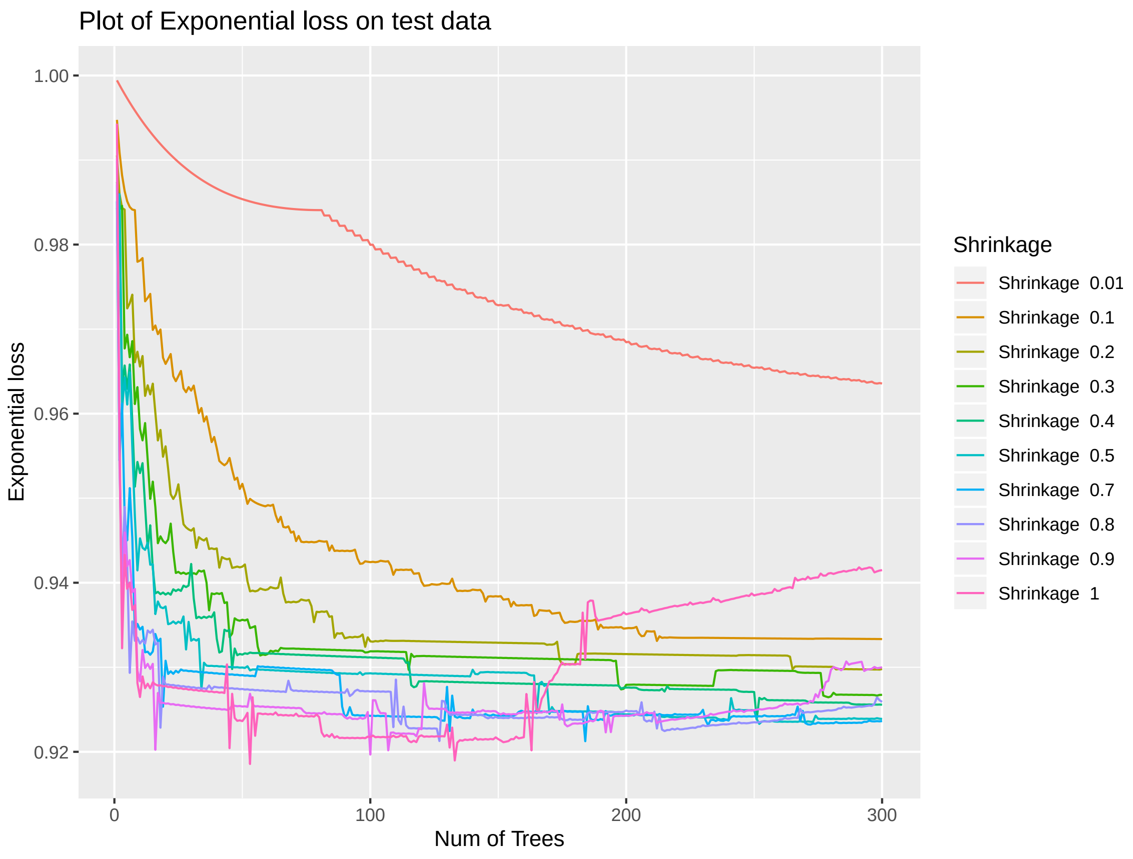
- Plot the final model (funtional value of F, and also the sign) with the observed data.
  - Below are the plots (for both training and test data) for final F
  - Data points are color coded based on the **predicted values Orange** : 1 and **Blue** : -1





- Exponential loss for test data:
  - Below is the plot for exponential-loss for test data, based on the fitted model (not sure if this is asked, but I have included it for completeness-sake).

– Loss is calculated in a similar way we calculate the loss when we were training the model (as here we have all the alphas, I just calculated the cumulative sum of the $\alpha * \delta * F(xtest)$ and then calculated the exponential loss after adding new tree to the model). This is done for the all the different models (with different Shrinkage δ parameter). *Code can be seen in RMD code file.*


Plot of Exponential loss on test data
Exponential loss
Num of Trees
Shrinkage
Shrinkage 0.01
Shrinkage 0.1
Shrinkage 0.2
Shrinkage 0.3
Shrinkage 0.4
Shrinkage 0.5
Shrinkage 0.7
Shrinkage 0.8
Shrinkage 0.9
Shrinkage 1