Supervised Learning Model Comparisn

Siddhesh Bagwe

2023-04-12

```
load("data_hw3_deepsolar.RData")
data1<-data
data1$solar_system_coverage <- ifelse(data1$solar_system_coverage=="high", 1, 0)
str(data)
```

```
## 'data.frame':    10926 obs. of  15 variables:
##  $ solar_system_coverage        : Factor w/ 2 levels "high","low": 1 1 1 1 1 1 1 1 1 1 ...
##  $ average_household_income     : num  78801 100499 73679 78841 83214 ...
##  $ employ_rate                  : num  0.915 0.895 0.914 0.943 0.907 ...
##  $ population_density           : num  4637 778 10832 1988 741 ...
##  $ housing_unit_count           : int  1512 3217 1906 2981 1985 2099 1876 1600 1384 2231
...
##  $ housing_unit_median_value    : num  325800 424000 339000 123500 220600 ...
##  $ occupancy_vacant_rate        : num  0.0278 0.0985 0.0866 0.0584 0.0368 ...
##  $ heating_fuel_gas_rate        : num  0.927 0.592 0.889 0.223 0.541 ...
##  $ heating_fuel_electricity_rate: num  0.0728 0.2776 0.0264 0.7738 0.4419 ...
##  $ heating_fuel_oil_rate        : num  0 0.00414 0.07467 0 0 ...
##  $ land_area                    : num  0.911 9.482 0.479 4.265 7.326 ...
##  $ water_area                   : num  0.00281 0 0.00352 4.22021 0 ...
##  $ air_temperature              : num  15.9 12.4 11.7 17.4 16.1 12.2 17.6 15.1 11.7 15.7
...
##  $ earth_temperature            : num  17.8 13.6 11.6 17.9 18.3 12.3 19.5 16.4 11.6 17 ...
##  $ daily_solar_radiation        : num  5.14 5.21 3.8 4.7 5.41 3.98 5.2 4.77 3.8 5.08 ...
```

The structure of the dataset shows us how our data is defined. We have 15 variables with 10926 observations with the solar_system_coverage as a factor and all the other variables as numerical and int. For ease of predictions we convert the categorical variable solar_system_coverage into numerical. Finally, for the model we will predict the solar_system_coverage using all the other parameters.

Logistic Regression

Logistic Regression is a type of supervised learning model used to predict the probability of an event happening. Here we will use it to predict the probability of solar_system_coverage being high or low.

```
set.seed(21200534)
fit <- glm(solar_system_coverage ~ ., data = data1, family = "binomial")
summary(fit)
```

```
##
## Call:
## glm(formula = solar_system_coverage ~ ., family = "binomial",
##     data = data1)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2405  -0.3187  -0.1972  -0.1002   3.9913
##
## Coefficients:
##                                Estimate Std. Error z value Pr(>|z|)
## (Intercept)                   -1.502e+01  1.296e+00 -11.592  < 2e-16 ***
## average_household_income      -7.407e-06  1.684e-06  -4.399 1.09e-05 ***
## employ_rate                   -4.082e+00  9.362e-01  -4.361 1.30e-05 ***
## population_density            -1.238e-04  7.975e-06 -15.526  < 2e-16 ***
## housing_unit_count             5.996e-04  4.898e-05  12.242  < 2e-16 ***
## housing_unit_median_value      4.343e-06  3.276e-07  13.258  < 2e-16 ***
## occupancy_vacant_rate         -8.515e+00  7.081e-01 -12.024  < 2e-16 ***
## heating_fuel_gas_rate          5.697e+00  7.833e-01   7.273 3.53e-13 ***
## heating_fuel_electricity_rate  3.732e+00  7.868e-01   4.743 2.11e-06 ***
## heating_fuel_oil_rate          9.907e+00  8.793e-01  11.268  < 2e-16 ***
## land_area                     -5.745e-03  1.322e-03  -4.346 1.39e-05 ***
## water_area                     2.615e-02  1.049e-02   2.494   0.0126 *
## air_temperature               -3.924e-01  8.257e-02  -4.752 2.01e-06 ***
## earth_temperature              3.262e-01  7.881e-02   4.139 3.49e-05 ***
## daily_solar_radiation          2.672e+00  1.725e-01  15.494  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7092.3  on 10925  degrees of freedom
## Residual deviance: 4284.0  on 10911  degrees of freedom
## AIC: 4314
##
## Number of Fisher Scoring iterations: 8
```

This is the output of a logistic regression model. The response variable is “solar_system_coverage”, which is binary (0 or 1), and the predictor variables are listed below it. The coefficients for each predictor variable are also listed, along with their standard errors, z-scores, and p-values.

The intercept term is -15.02, and all predictor variables are standardized. For example, a one-unit increase in “average_household_income” corresponds to a 7.4e-06 decrease in the log-odds of having solar system coverage, with no change in other parameters.

Next, we will check how well the logistic regression model predicts on our data. we know that, Logistic regression predicts the value in the range [0,1], so we need to decide the threshold(tau) to check whether solar_system_coverage is low or high.

```
tau <- 0.5
p <- fitted(fit)
pred <- ifelse(p > tau, 1, 0)
# cross tabulation between observed and predicted
tab <- table(data1$solar_system_coverage, pred)
tab
```

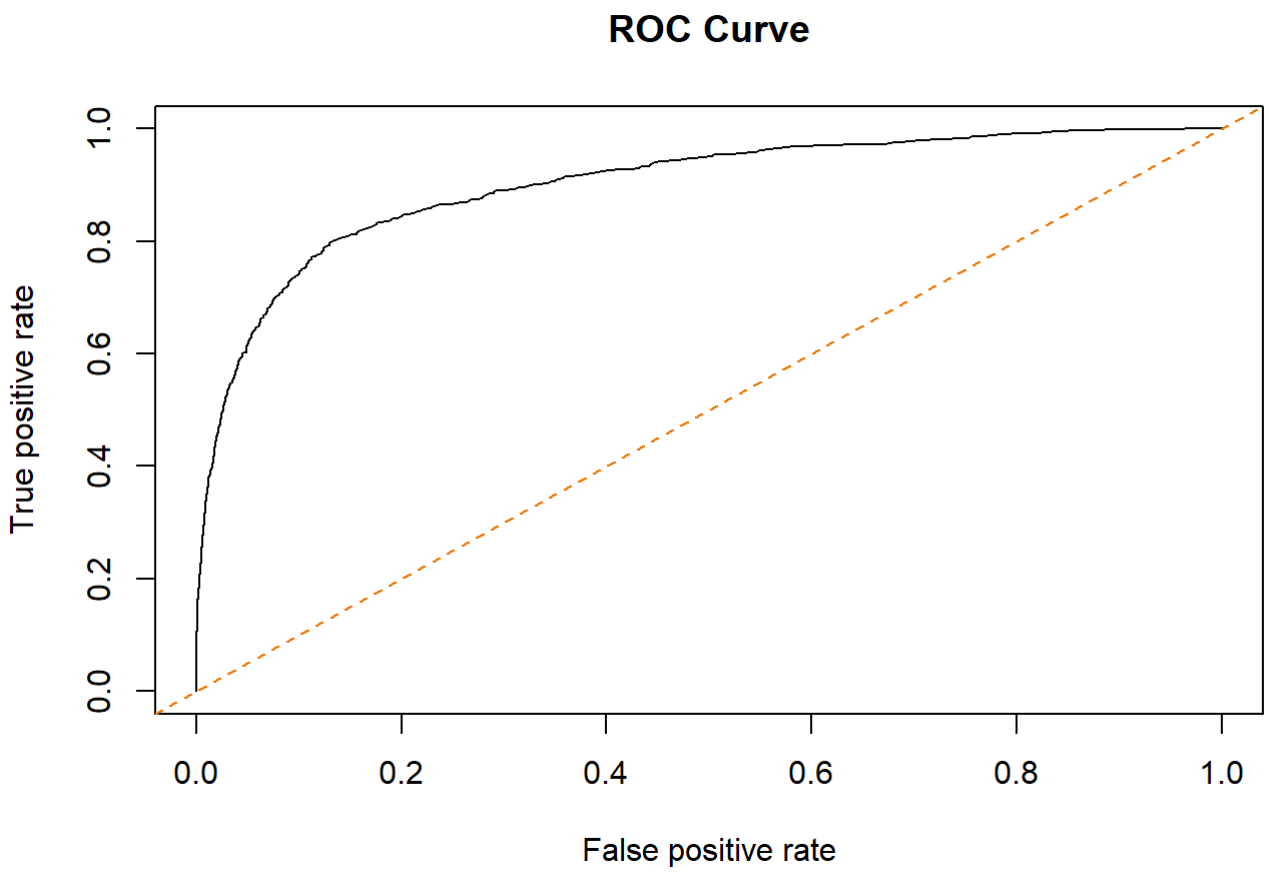
```
##    pred
##       0    1
##   0 9653  183
##   1  611  479
```

```
sum(diag(tab))/sum(tab)
```

```
## [1] 0.9273293
```

First we put the value of tau = 0.5 where value>0.5 gives the value of solar_system_coverage as 1(high) and low (0) when tau<0.5. Using this value we check the performance of our model. from the table above. We see that the model predicts with an accuracy of 0.9273293 which is a good accuracy however, the tau we chose is just a guess. We can use the ROC and PCR curves to find the optimal values of tau.

```
library(ROCR)
pred_obj <- prediction(fitted(fit), data1$solar_system_coverage)
roc <- performance(pred_obj, "tpr", "fpr")
plot(roc,main="ROC Curve")
abline(0, 1, col = "darkorange2", lty = 2 ) # add bisect line
```

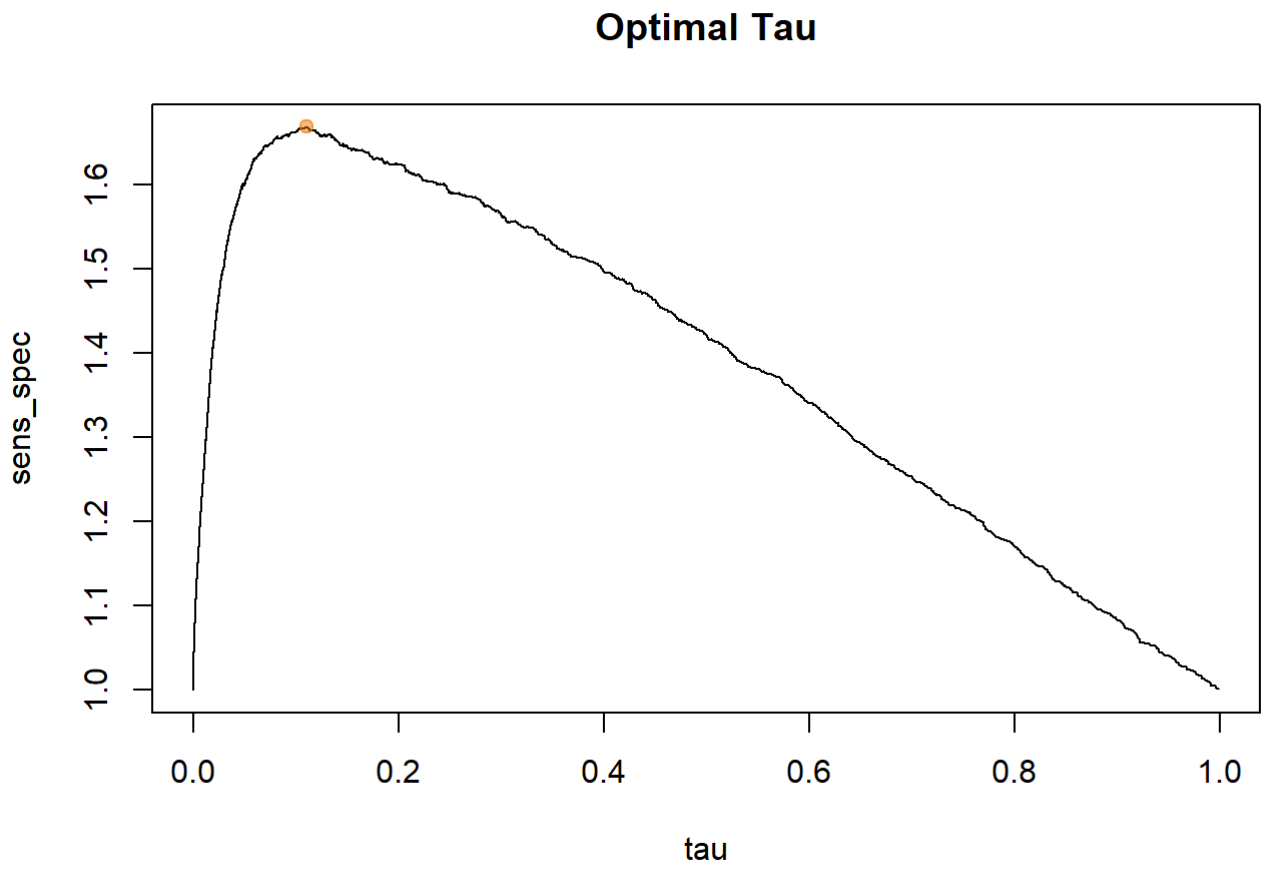



```
# compute the area under the ROC curve
auc <- performance(pred_obj, "auc")
auc@y.values
```

```
## [[1]]
## [1] 0.9006681
```

The plot above shows the ROC curve for our fit. The ROC curve is a plot of true positive rate v/s the false positive rate. The AUC (Area under ROC curve) measures the entire two-dimensional area underneath the entire ROC curve. AUC provides an aggregate measure of performance across all possible classification thresholds.So AUC can provide a measure of the accuracy of the model. The AUC calculated here is 0.9006681. From the model performance, we can find the optimal tau using the sensitivity and specificity from the fit.

```
sens <- performance(pred_obj, "sens")
spec <- performance(pred_obj, "spec")
tau <- sens@x.values[[1]]
sens_spec <- sens@y.values[[1]] + spec@y.values[[1]]
best_roc <- which.max(sens_spec)
plot(tau, sens_spec, type = "l", main ="Optimal Tau")
points(tau[best_roc], sens_spec[best_roc], pch = 19, col = adjustcolor("darkorange2", 0.5))
```




```
tau[best_roc] # optimal tau according to the ROC curve
```

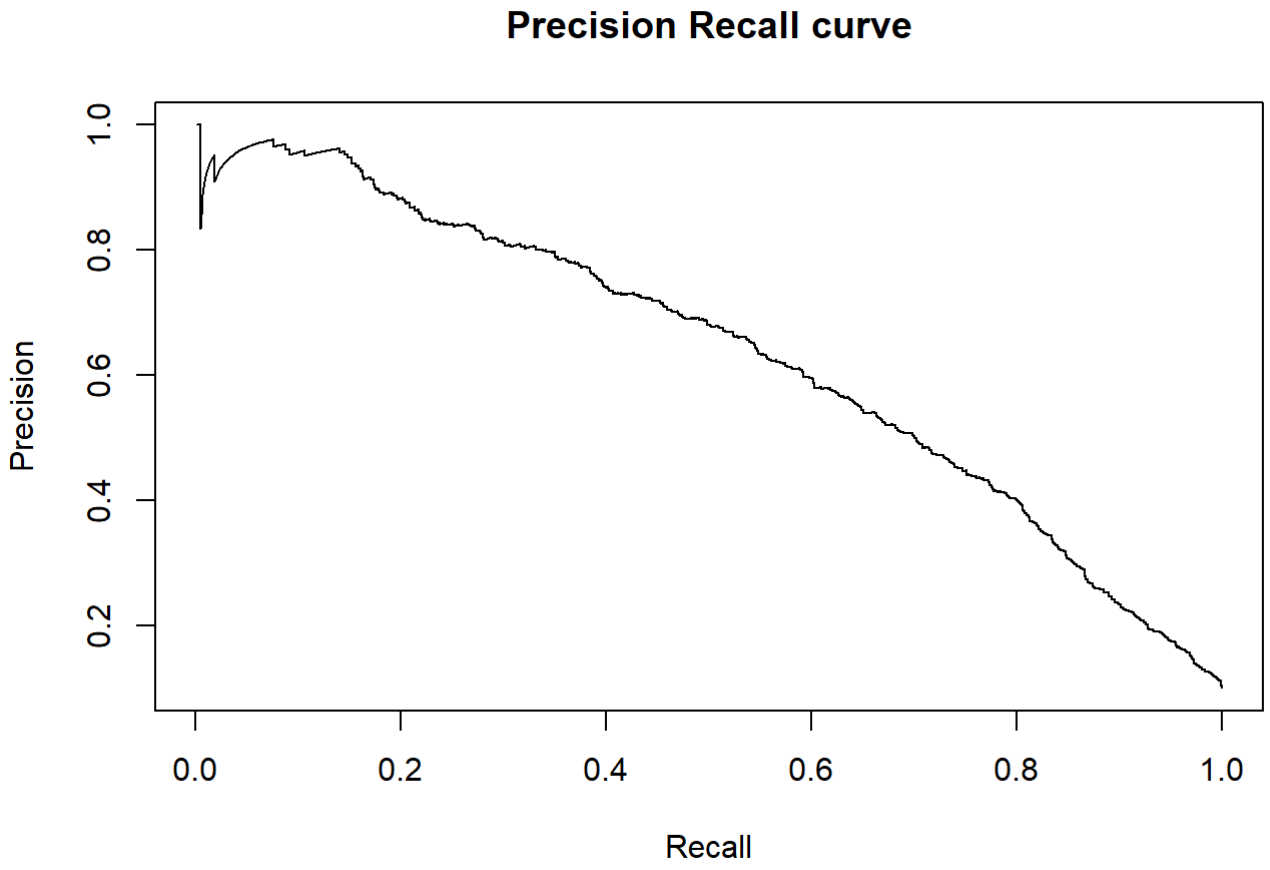
```
##      8383
## 0.1104322
```

```
# classification for optimal tau
pred <- ifelse(fitted(fit) > tau[best_roc], 1, 0)
tab <- table(data1$solar_system_coverage, pred)
sum(diag(tab))/sum(tab)
```

```
## [1] 0.8610653
```

Using the specificity and sensitivity we get the value of optimal tau = 0.1104322. Using this value as threshold we get the predication accuracy as 0.8610653 which is worse than the one with tau = 0.5. So we go ahead with the PR curve to get the optimal tau value.

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

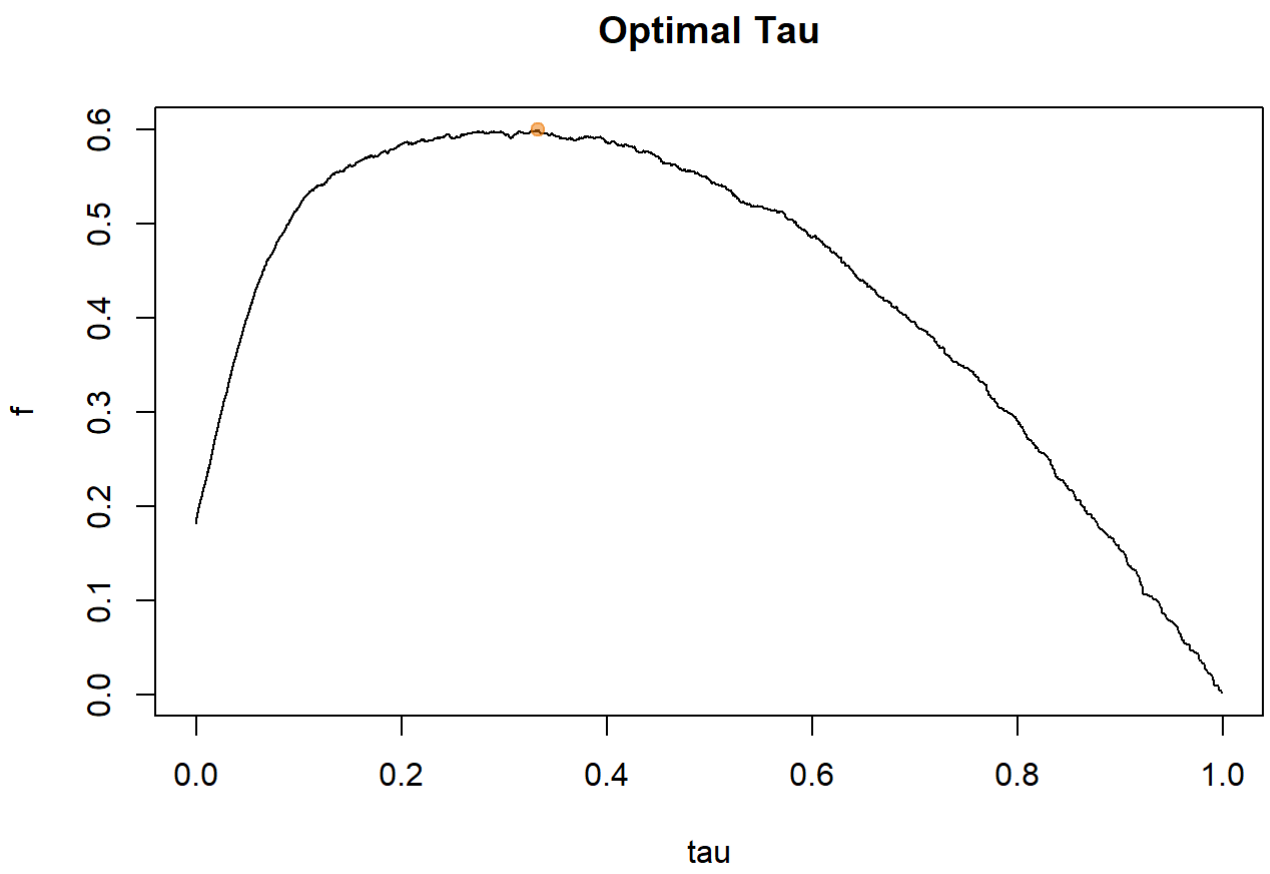



```
# compute area under the PR curve
aucpr <- performance(pred_obj, "aucpr")
aucpr@y.values
```

```
## [[1]]
## [1] 0.6353418
```

Just like the ROC curve, The PR curve is also used to assess the model performance.We get the PR curve by plotting the recall vs precision for the data at various thresholds. The AUC(area under the curve) for the PR curve can also give the accuracy of the model with 0 being the worst and 1 being the best. For the given model, the AUC of the PR curve is 0.6353418.

```
perf_f <- performance(pred_obj, "f")
tau <- perf_f@x.values[[1]]
f <- perf_f@y.values[[1]]
best_pr <- which.max(f)
plot(tau, f, type = "l", main = "Optimal Tau")
points(tau[best_pr], f[best_pr], pch = 19, col = adjustcolor("darkorange2", 0.5))
```




```
tau[best_pr] # optimal tau according to the PR curve
```

```
##    11424
## 0.332528
```

```
# classification for optimal tau
pred <- ifelse(fitted(fit) > tau[best_pr], 1, 0)
tab <- table(data1$solar_system_coverage, pred)
tab
```

```
##    pred
##       0    1
##   0 9426  410
##   1  449  641
```

```
sum(diag(tab))/sum(tab)
```

```
## [1] 0.9213802
```

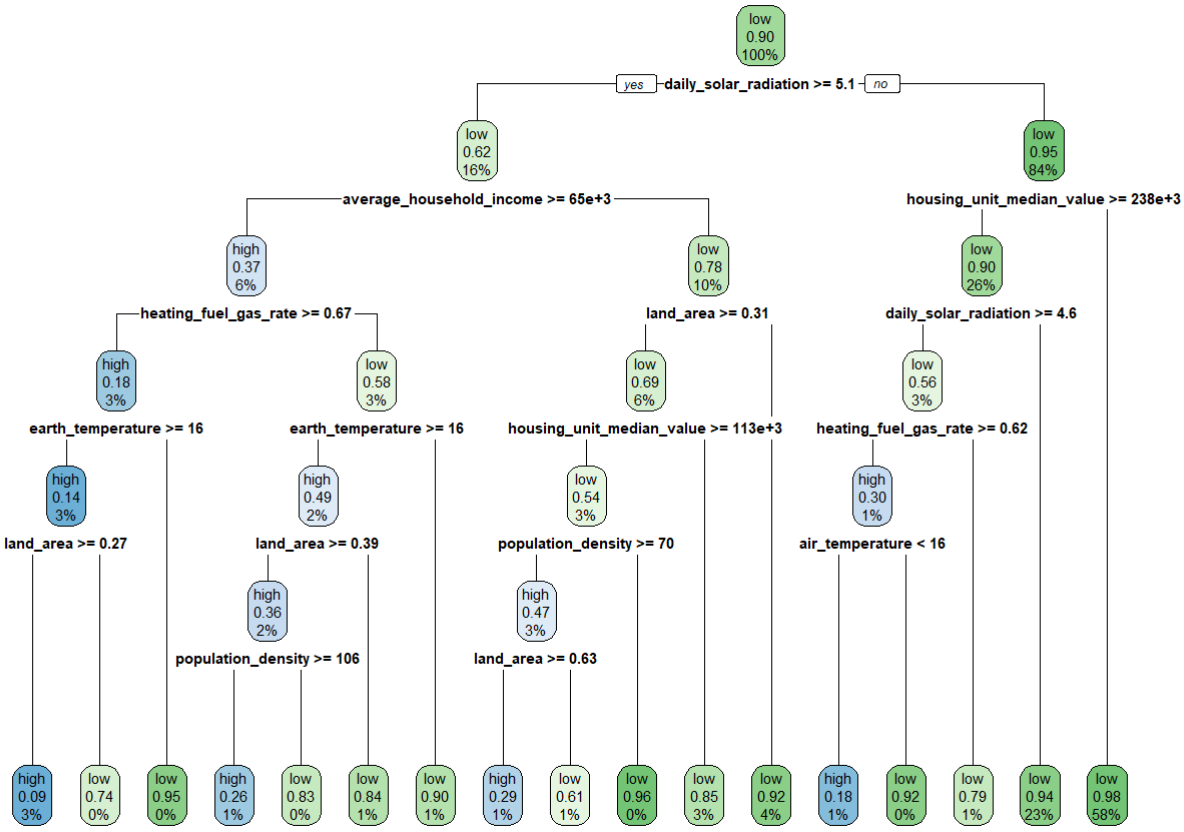
Using the f value from the PR curve, we get the optimal tau from the point where the f values are highest. We get the optimal tau as 0.332528. Using this value of tau to as the threshold for prediction we get the above correlation matrix. The accuracy is found to be 0.9213802.

From the different thresholds calculated, we see that the threshold tau=0.5 gives us the best accuracy.

Classification Tree

Classification Tree Analysis (CTA) is an analytical procedure that takes examples of known classes (i.e., training data) and constructs a decision tree based on measured attributes such as reflectance. The splits help us decide the classes of the output (high or low here)

```
library(rpart)
library(rpart.plot)
set.seed(21200534)
ct <- rpart(solar_system_coverage ~ ., data = data)
rpart.plot(ct)
```



Here, we have fit a classification tree model for our dataset. The data is trained for n= 10926 parameters. Each node in the tree represents a decision based on a specific feature or set of features, and the model splits the data into different groups based on these decisions in order to make predictions. The model is made up of a series of nodes that represent decisions based on input features, leading to either a predicted outcome or further decisions. The text lists the nodes of the model, each with its own split criterion, number of observations at that node, the calculated loss function, the predicted outcome value, and the probabilities associated with each outcome value.

```
phat <- predict(ct)
# predicted classes can be obtained in a similar way
# by using the argument "type"
class <- predict(ct, type = "class")
tab<-table(data$solar_system_coverage, class)
tab
```

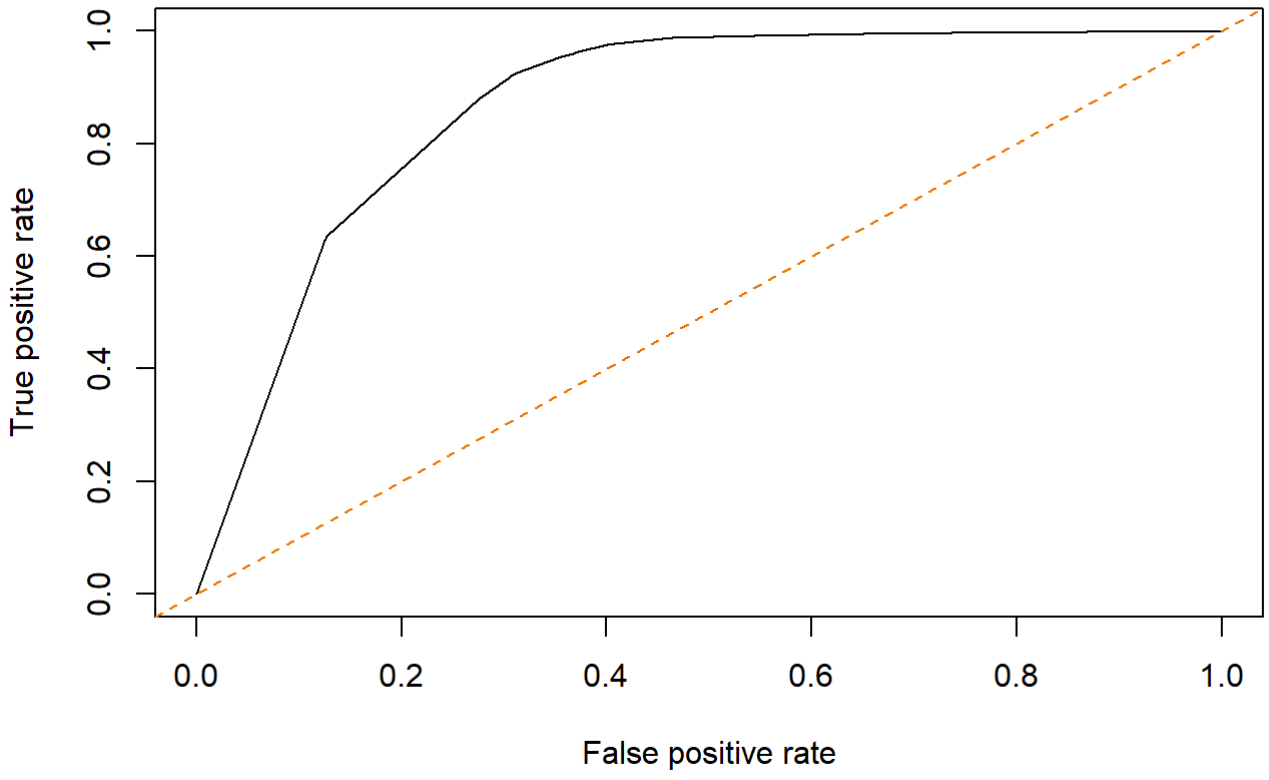
```
##       class
##        high  low
##   high  584  506
##   low   123 9713
```

```
sum(diag(tab))/sum(tab)
```

```
## [1] 0.9424309
```

Using the classification we assess the model performance in predicting the given data. The confusion matrix shows how many times the model succeeds in predicting the true values. The accuracy in this case is found to be 0.9424309

```
pred_obj <- prediction(phat[,2], data$solar_system_coverage)
roc <- performance(pred_obj, "tpr", "fpr")
plot(roc)
abline(0,1, col = "darkorange2", lty = 2)
```

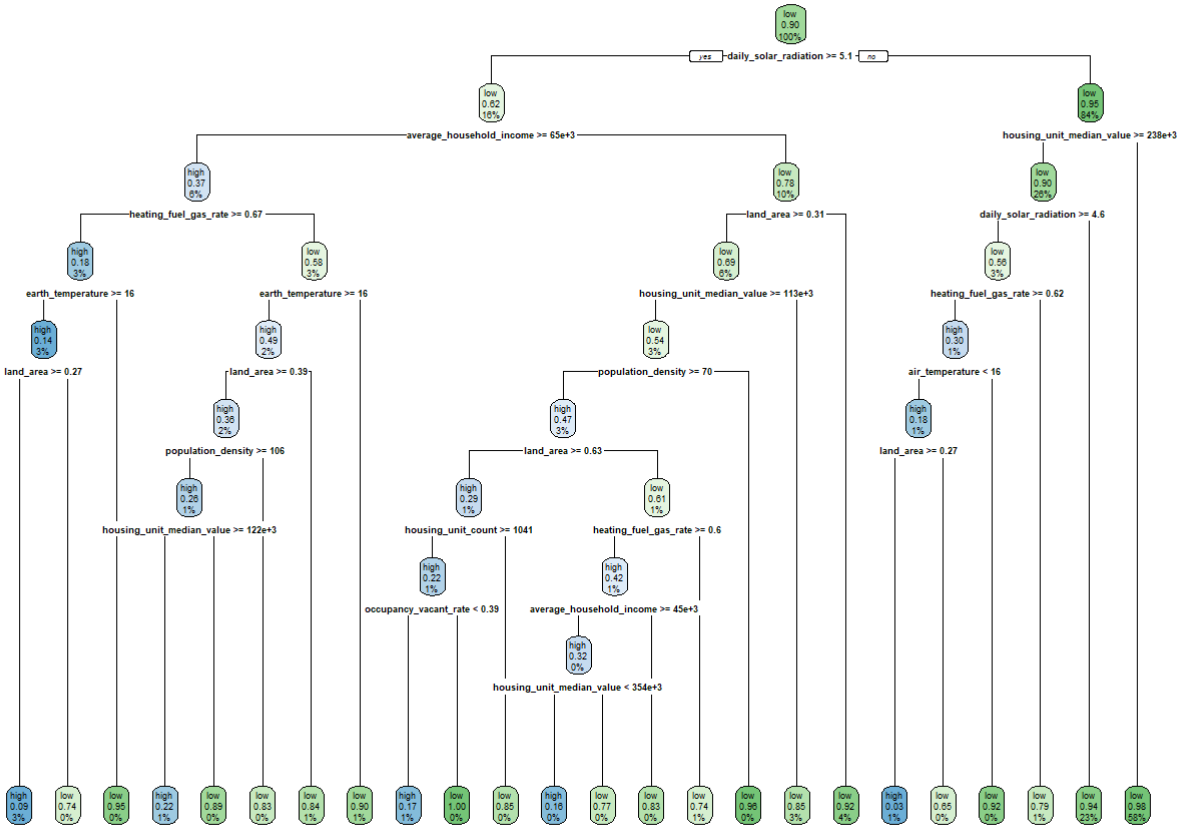


```
# compute the area under the ROC curve
auc <- performance(pred_obj, "auc")
auc@y.values
```

```
## [[1]]
## [1] 0.8664965
```

The above plot shows the ROC curve for the classification tree model that we fit. The AUC value here 0.8664965 which is a good value for the AUC. Although this model does a good job, we can make few changes to hyperparameters to improve the fit. We will now try to tweak around with certain parameters to check whether it improves the model or not.

```
set.seed(21200534)
ct3 <- rpart(solar_system_coverage ~ ., data = data, cp = 0.01/2, minsplit = 2)
rpart.plot(ct3)
```



For the next model, we introduce the hyperparameters to the model.

1. minsplit - Set the minimum number of observations in the node before the algorithm perform a split
2. complexity parameter (cp) - The complexity parameter (cp) in rpart is the minimum improvement in the model needed at each node.

For this model, we set the minsplit as 2, that there needs to be atleast 2 observations in any node before making any splits. We can see this in the plot above. The complexity parameter is set to 0.01/2. This helps in speeding up the splitting process. We fit the model using this hyperparameters for our data.

```
phat3 <- predict(ct3)
class3 <- predict(ct3, type = "class")
tab<-table(data$solar_system_coverage, class3)
tab
```

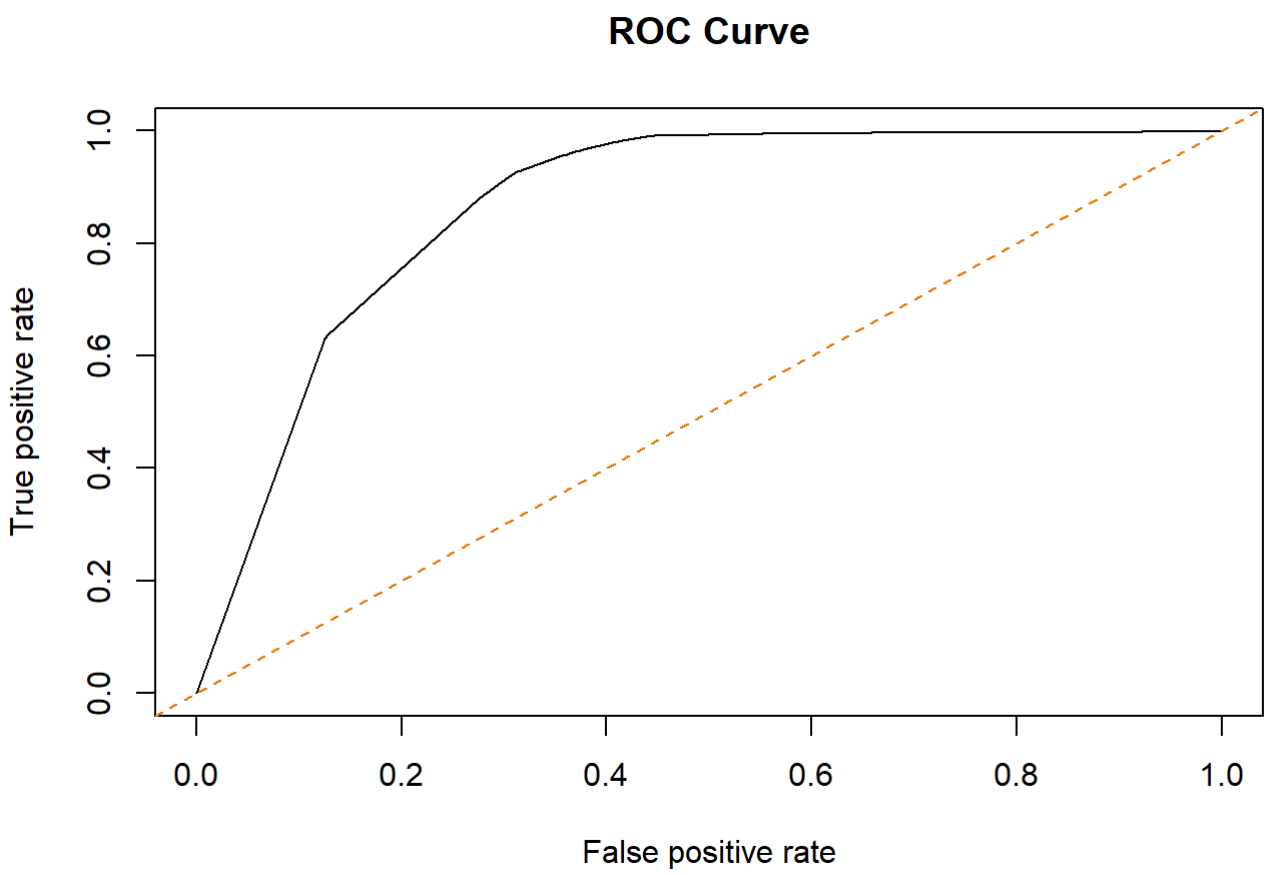
```
##       class3
##        high  low
##   high  603  487
##   low    86 9750
```

```
sum(diag(tab))/sum(tab)
```

```
## [1] 0.9475563
```

The confusion matrix using this classification tree is seen above. We get an accuracy of 0.9475563 which is a slight improvement on the model before.

```
pred_obj3 <- prediction(phat3[,2], data$solar_system_coverage)
roc <- performance(pred_obj3, "tpr", "fpr")
plot(roc,main="ROC Curve")
abline(0,1, col = "darkorange2", lty = 2)
```




```
# compute the area under the ROC curve
auc3 <- performance(pred_obj3, "auc")
auc3@y.values
```

```
## [[1]]
## [1] 0.8679009
```

The plot above shows the ROC curve for the model. The AUC for this model is 0.8679009 which shows that this model is better than the one before.

```
set.seed(21200534)
ct4 <- rpart(solar_system_coverage ~ ., data = data, cp = 0.01/10, minsplit = 1)
phat4 <- predict(ct4)
class4 <- predict(ct4, type = "class")
tab<- table(data$solar_system_coverage, class4)
tab
```

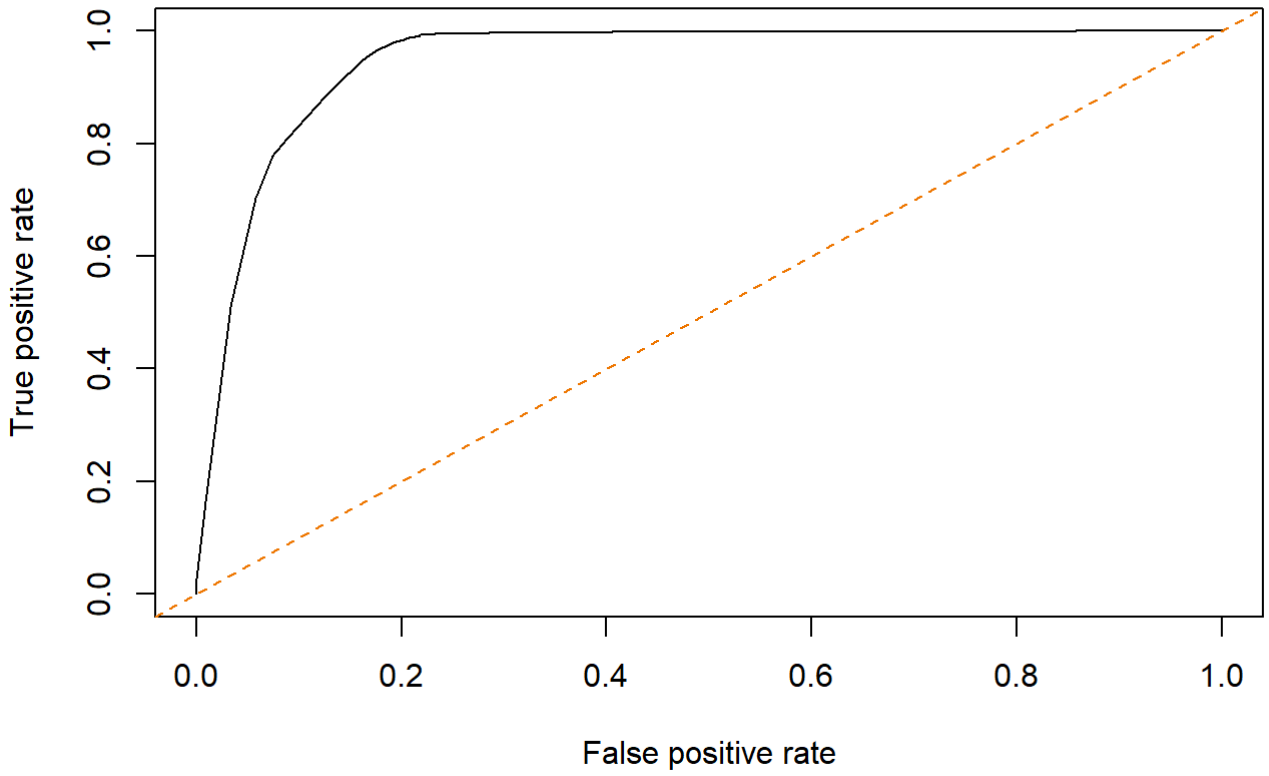
```
##       class4
##        high  low
##   high  848  242
##   low    58 9778
```

```
sum(diag(tab))/sum(tab)
```

```
## [1] 0.9725426
```

Next, we try to make the model more complex by reducing the complexity parameter. Also we change the minsplit to 1, so there are no restrictions for the split. Using this model, we calculate the accuracy of the fit to be 0.9725426 which is the best accuracy for the classification tree we have observed. We will try to check if this is the best model using the ROC curve.

```
pred_obj4 <- prediction(phat4[,2], data$solar_system_coverage)
roc <- performance(pred_obj4, "tpr", "fpr")
plot(roc)
abline(0,1, col = "darkorange2", lty = 2)
```




```
# compute the area under the ROC curve
auc4 <- performance(pred_obj4, "auc")
auc4@y.values
```

```
## [[1]]
## [1] 0.9480075
```

The ROC curve plot shows that the model does a good job in predicting the true positives. The AUC value of 0.9480075 is also the best accuracy for any classification tree model.

Random forest

Random forest is another supervised learning method which we can employ for the dataset. It builds decision trees on different samples and takes their majority vote for classification.

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(21200534)
# implement the random forest algorithm
fit_rf <- randomForest(solar_system_coverage ~ ., data = data, importance = TRUE)
# examine the results
fit_rf
```

```
##
## Call:
##  randomForest(formula = solar_system_coverage ~ ., data = data,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 5.1%
## Confusion matrix:
##      high  low class.error
## high  631  459   0.4211009
## low    98 9738   0.0099634
```

The summary above shows the model of random forest. The model uses 500 decision trees, and at each split in the tree, only 3 variables were considered for splitting. The OOB estimate rate is 5.1% which shows that the model has an error of 5% while predicting the data. The confusion matrix further shows the TP, TN, FP, FN rates for the model. We will now check the model's predictive performance.

```
# check predictions
pred_rf <- predict(fit_rf, type = "class")
tab <- table(data$solar_system_coverage, pred_rf)
tab
```

```
##       pred_rf
##        high  low
##   high  631  459
##   low    98 9738
```

```
sum(diag(tab))/sum(tab)
```

```
## [1] 0.9490207
```

Next we try predictions using the random forest model to check the accuracy of the model. The confusion matrix shows how the model predicts the high and low value of solar_system_coverage. The accuracy calculated for this model is 0.9490207 which shows that the model does a good job while predicting the data.

Support Vector Machines

Support Vector Machines is another supervised learning method which can be used both for regression and classification. In classification, it uses the support vectors to separate the data into classes.

```
library(kernlab)
x <- scale(as.matrix(data[,2:15]))
y <- data[,1]
```

For fitting an svm model, the kvsm function requires the input data in matrix form, so we scale our data to matrix form.

```
set.seed(21200534)
svm_poly_1 <- ksvm(x, y, type = "C-svc",
kernel = "polydot", kpar = list(degree = 2))
svm_poly_2 <- ksvm(x, y, type = "C-svc",
kernel = "polydot", kpar = list(degree = 3))
svm_grbf_1 <- ksvm(x, y, type = "C-svc",
kernel = "rbfdot", kpar = list(sigma = 0.5))
svm_grbf_2 <- ksvm(x, y, type = "C-svc",
kernel = "rbfdot", kpar = list(sigma = 1.5))
svm_lrbf <- ksvm(x, y, type = "C-svc",
kernel = "rbfdot", kpar = "automatic")
svm_tanh <- ksvm(x, y, type = "C-svc",
kernel = "tanhdot", kpar = list(scale = 0.5))
```

We have tried fitting svm using various combinations to find the best fit for the model. The different svm's used are

- The first two SVM models use polynomial kernels with degrees of 2 and 3, respectively. This model computes the dot product between the 2 vectors upto the given degree
- The third and fourth SVM models use GRBF (Gaussian radial basis function) kernels with different values of sigma, which controls the width of the Gaussian function.
- The fifth SVM model uses LRBF (linear radial basis function) kernel with sigma set to automatic.
- The final SVM model is the sigmoid maodel with a scale of 0.5

We will use this models to get the prediction and decide which model is the best fit for our data.

```
class_svm1 <- predict(svm_poly_1)
class_svm2 <- predict(svm_poly_2)
class_svm3 <- predict(svm_grbf_1)
class_svm4 <- predict(svm_grbf_2)
class_svm5 <- predict(svm_lrbf)
class_svm6 <- predict(svm_tanh)

tab_svm1 = table(data$solar_system_coverage, class_svm1)
sum(diag(tab_svm1))/ sum(tab_svm1)
```

```
## [1] 0.9471902
```

```
tab_svm2 = table(data$solar_system_coverage, class_svm2)
sum(diag(tab_svm2))/ sum(tab_svm2)
```

```
## [1] 0.9610104
```

```
tab_svm3 = table(data$solar_system_coverage, class_svm3)
sum(diag(tab_svm3))/ sum(tab_svm3)
```

```
## [1] 0.9621087
```

```
tab_svm4 = table(data$solar_system_coverage, class_svm4)
sum(diag(tab_svm4))/ sum(tab_svm4)
```

```
## [1] 0.9788578
```

```
tab_svm5 = table(data$solar_system_coverage, class_svm5)
sum(diag(tab_svm5))/ sum(tab_svm5)
```

```
## [1] 0.9467326
```

```
tab_svm6 = table(data$solar_system_coverage, class_svm6)
sum(diag(tab_svm6))/ sum(tab_svm6)
```

```
## [1] 0.8192385
```

From the predictions we see that the model with Gaussian radial basis function and sigma 1.5 performs the best for our data with an accuracy of 0.9788578.

Parameter Tuning to predict the best model

To select the best model from all the above methods. We use the best models found in each of the logistic regression, classification tree, random forest and SVM models.

```
# we use this function to compute performance metrics
class_metrics <- function(y, yhat) {
tab <- table(y, yhat)
c( sum(diag(tab))/sum(tab), # accuracy
tab[2,2]/sum(tab[2,]), # sensitivity
tab[1,1]/sum(tab[1,]) # specificity
)
}
# set number of replicates and containers
N <- nrow(data)
R <- 10 #
# store results
# each slice corresponds to a metric
out_test <- out_val <- out_train <- array(NA, dim = c(R, 4, 3))
```

The first step is to create the accuracy parameters which would be used to compare all the models. This is defined in the class_metrics function. We define the accuracy, specificity and sensitivity for the predictions in the function.

```
for ( r in 1:R ) {
  # a 70-15-15 split
  train <- sample(1:N, 0.7*N)
  val <- sample(setdiff(1:N, train), (N - 0.7*N)*0.5)
  test <- setdiff(1:N, c(train, val))

  #  fit a logistic regression model
  fit_lr <- glm(solar_system_coverage ~ ., data = data1, family = "binomial", subset = train)

  #  fit a single classification tree
  fit_ct <- rpart(solar_system_coverage ~ ., data = data, cp = 0.01/10, minsplit = 1,subset =
train)

  # random forest
  fit_rf <- randomForest(solar_system_coverage ~ ., data = data, subset = train, importance =
TRUE)

  # Support Vector machine
  fit_svm <- ksvm(x, y, type = "C-svc",kernel = "rbfdot", kpar = list(sigma = 1.5))

  # Predicting on training data
  pred_lr <- ifelse(predict(fit_lr, newdata = data1[train,], type = "response") > tau[best_p
r], 1, 0)
  out_train[r,1,]<-class_metrics(data1$solar_system_coverage[train], pred_lr)

  pred_ct <- predict(fit_ct, newdata = data[train,], type = "class")
  out_train[r,2,] <- class_metrics(data$solar_system_coverage[train], pred_ct)

  pred_rf <- predict(fit_rf, newdata = data[train,], type = "class")
  out_train[r,3,] <- class_metrics(data$solar_system_coverage[train], pred_rf)

  pred_svm <- predict(fit_svm, newdata = x[train,])
  out_train[r,4,] <- class_metrics(data$solar_system_coverage[train], pred_svm)

  # Predicting on validation data
  pred_lr <- ifelse(predict(fit_lr, newdata = data1[val,], type = "response") > 0.5, 1, 0)
  out_val[r,1,]<-class_metrics(data1$solar_system_coverage[val], pred_lr)

  pred_ct <- predict(fit_ct, newdata = data[val,], type = "class")
  out_val[r,2,] <- class_metrics(data$solar_system_coverage[val], pred_ct)

  pred_rf <- predict(fit_rf, newdata = data[val,], type = "class")
  out_val[r,3,] <- class_metrics(data$solar_system_coverage[val], pred_rf)

  pred_svm <- predict(fit_svm, newdata = x[val,])
  out_val[r,4,] <- class_metrics(data$solar_system_coverage[val], pred_svm)

    # Predicting on test data
  pred_lr <- ifelse(predict(fit_lr, newdata = data1[test,], type = "response") > tau[best_p
r], 1, 0)
  out_test[r,1,]<-class_metrics(data1$solar_system_coverage[test], pred_lr)

  pred_ct <- predict(fit_ct, newdata = data[test,], type = "class")
  out_test[r,2,] <- class_metrics(data$solar_system_coverage[test], pred_ct)
```

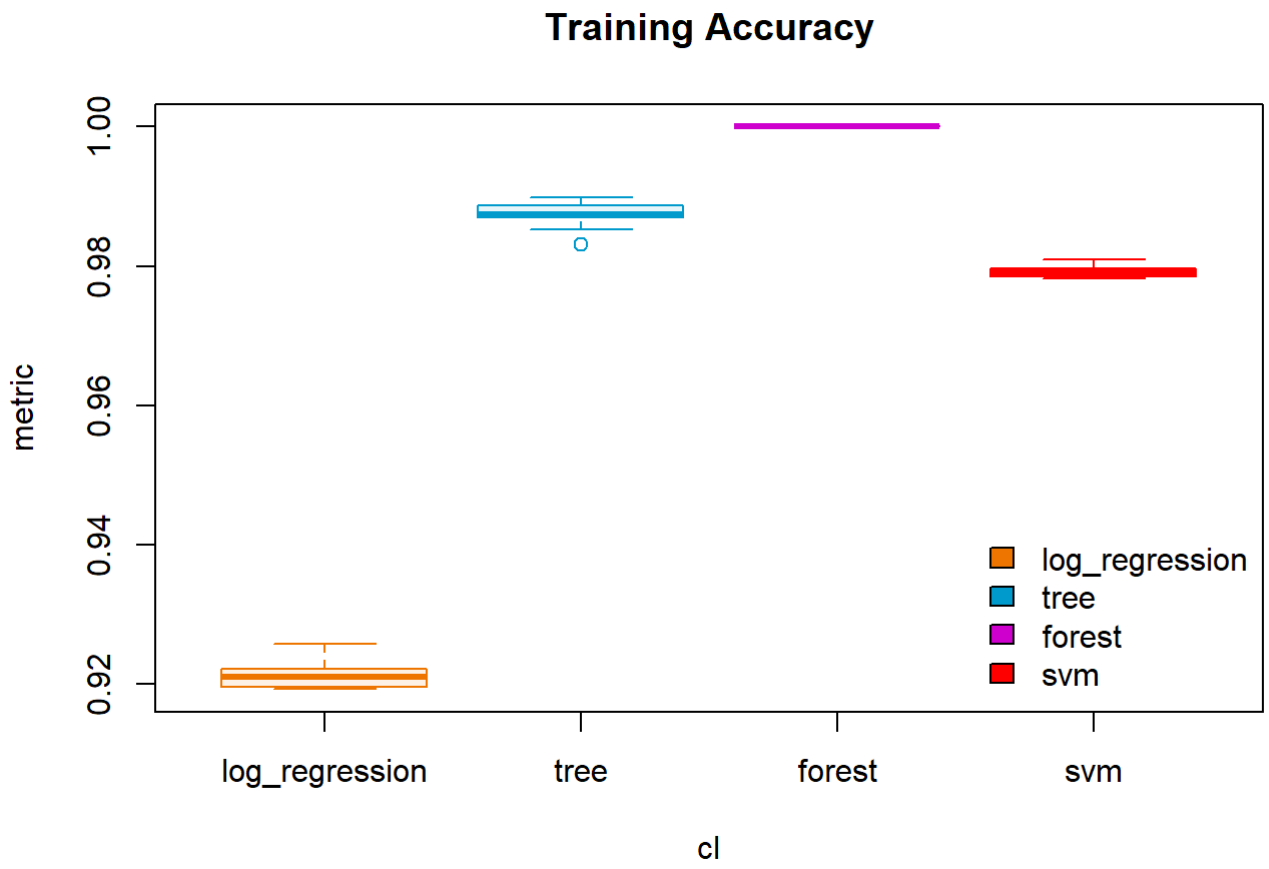
```
  pred_rf <- predict(fit_rf, newdata = data[test,], type = "class")
  out_test[r,3,] <- class_metrics(data$solar_system_coverage[test], pred_rf)

  pred_svm <- predict(fit_svm, newdata = x[test,])
  out_test[r,4,] <- class_metrics(data$solar_system_coverage[test], pred_svm)
}
```

For comparing the models, we run the the entire process of model fitting and predictions R number of times here(10 to have effective computing time). We fit the models with the train data and run predictions on the train, validation and test data. After running the loop for R times we get 10 different predictions. Using the class_metrics function we calculate the accuracy, specificity and sensitivity

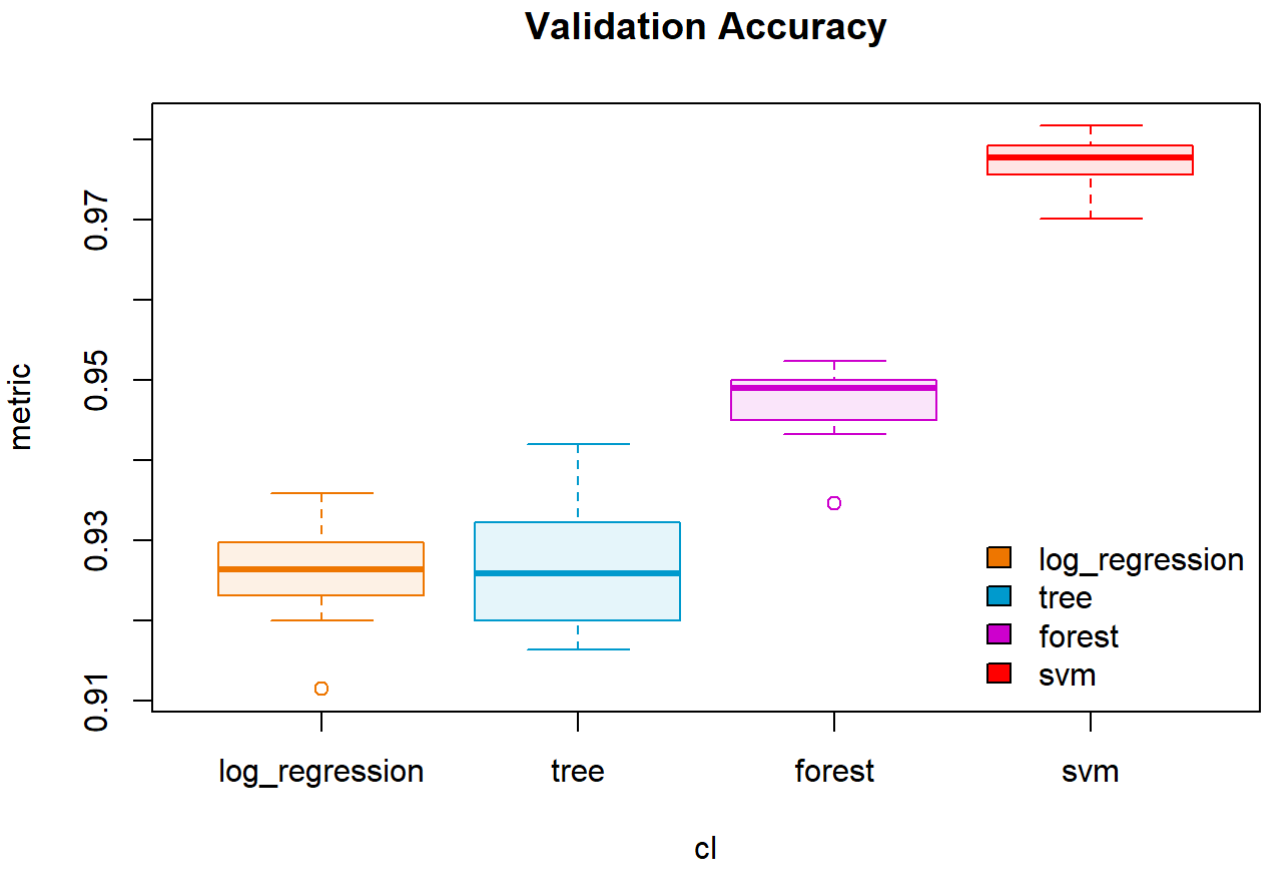
```
metric <- c(out_train[,,1])
cl <- factor(rep(1:4,each = R), labels = c("log_regression","tree", "forest","svm"))
metric <- data.frame(metric = metric, cl = cl)
# colors denote classifiers
cols <- c("darkorange2", "deepskyblue3", "magenta3", "red")
# 'fancy' boxplot to show all results in a single panel
boxplot(metric ~ cl , data = metric,
border = cols, col = adjustcolor(cols, 0.1),main ="Training Accuracy")

legend("bottomright", legend = levels(cl), bty = "n", fill = cols)
```



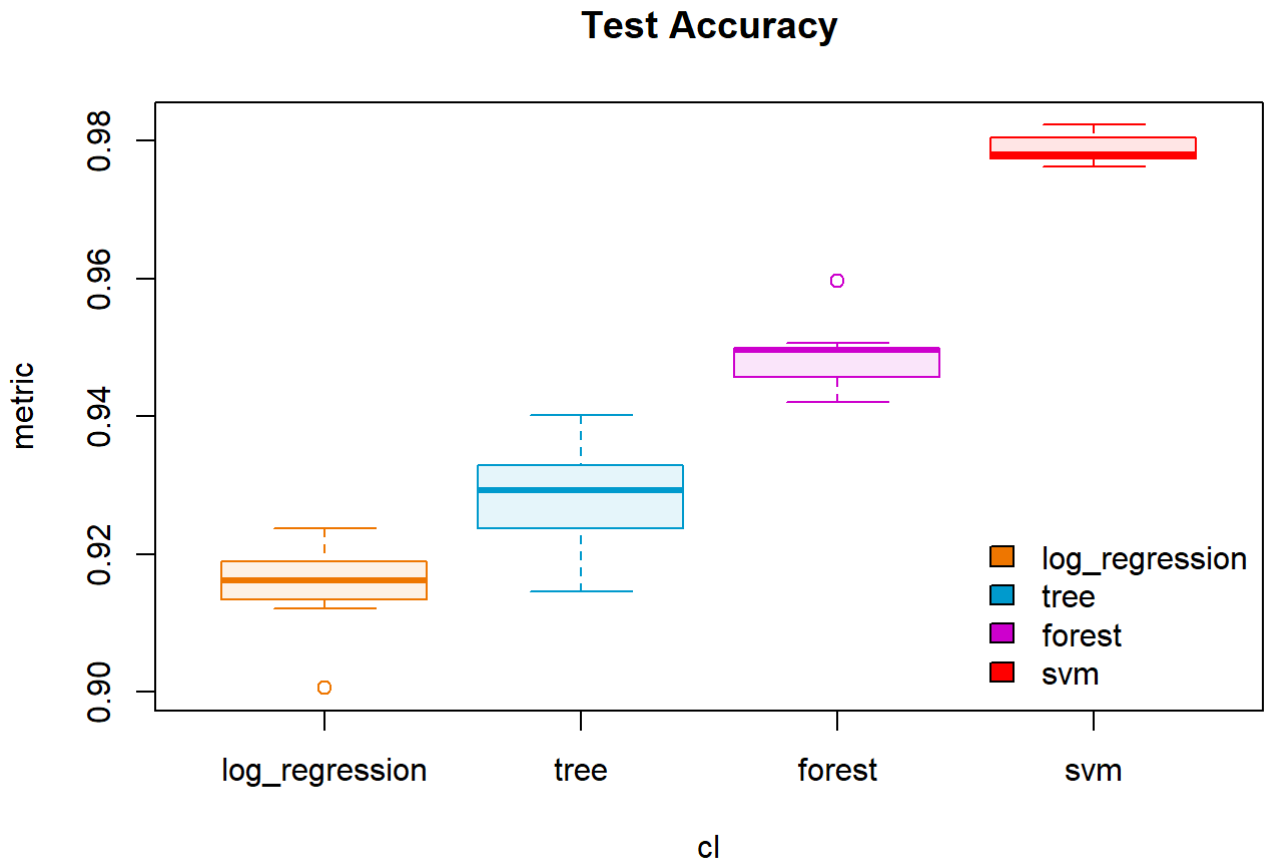
```
metric <- c(out_val[,,1])
cl <- factor(rep(1:4, each = R), labels = c("log_regression","tree", "forest","svm"))
metric <- data.frame(metric = metric, cl = cl)
# colors denote classifiers
cols <- c("darkorange2", "deepskyblue3", "magenta3", "red")
# 'fancy' boxplot to show all results in a single panel
boxplot(metric ~ cl , data = metric,
border = cols, col = adjustcolor(cols, 0.1), main ="Validation Accuracy")

legend("bottomright", legend = levels(cl), bty = "n", fill = cols)
```




```
metric <- c(out_test[,,1])
cl <- factor(rep(1:4, each = R), labels = c("log_regression","tree", "forest","svm"))
metric <- data.frame(metric = metric, cl = cl)
# colors denote classifiers
cols <- c("darkorange2", "deepskyblue3", "magenta3","red")
# 'fancy' boxplot to show all results in a single panel
boxplot(metric ~ cl , data = metric,
border = cols, col = adjustcolor(cols, 0.1),main ="Test Accuracy")

legend("bottomright", legend = levels(cl), bty = "n", fill = cols)
```



Using the accuracy parameter, we plot the train, validation and test accuracy to check which model works the best for our data. The plot analysis is as below.

Training Accuracy - We see that the Random Forest works best for the training data and gives the highest accuracy followed by Classification tree and SVM with logistic regression being the worst.

Validation Accuracy - For the validation data, which is a new data for the model, SVM performs considerably better than all the other models followed by random forest, classification tree and logistic regression in that order.

Test Accuracy - The test data is also predicted the best by SVM, Random forest is the second best followed by Classification tree and Logistic Regression.

By the analysis, it is clear that our SVM model does the best job in predicting the solar_system_coverage for the new data.

Prediction on Test Data

Finally, we will use the best model (SVM) to predict a new test data. Here we won't use any validation data and use 80% of data to train the model.. Next we fit the svm model.

```
set.seed(21200534)
train <- sample(1:nrow(data), N*0.8)
test <- setdiff(1:N, train)
x_train = x[train,]
y_train = y[train]
x_test = x[test,]
y_test = y[test]

svm <- ksvm(x_train, y_train, type = "C-svc",kernel = "rbfdot", kpar = list(sigma = 1.5))
```

```
pred_svm <- predict(svm, newdata = x_test)
tab = table(y_test, pred_svm)
sum(diag(tab))/sum(tab)
```

```
## [1] 0.9304666
```

```
tab
```

```
##       pred_svm
## y_test high  low
##   high   76  141
##   low    11 1958
```

Finally using the SVM model we can see the confusion matrix of the predictions. The model predicts solar_system_coverage as ‘high’ accurately 76 times and wrong 141 times. while ‘low’ accurately 1958 times and wrong 11 times which gives us a accuracy of 0.9304666.