# Датасет

**drones preprocessed\_classified (2).csv**

|  |  |  |  |
| --- | --- | --- | --- |
| Naïve Bayes | Random Forest | K-Nearest Neighbors | Support-vector machines |
| **0.882** | **0.736** | **0.9324** | **0.9396** |

# Pipeline

data <- read.csv(choose.files(), header = T, sep = ";", stringsAsFactors = T)

>

> work.data <- data[c('Publication.Year', 'Count.of.Simple.Family.Members', 'Simple.Family.Cited.by.Count',

+ 'Count.of.Other.References', 'Count.of.Cited.by.Patents', 'Count.of.Cites.Patents', 'Count.of.Cited.by.Patents.Within.3.years',

+ 'Count.of.Cited.by.Patents.Within.5.years', 'Simple.Legal.Status', 'Count.of.claims', 'Quality.of.Family',

+ 'Classifier')]

> work.data$Publication.Year <- 2020-work.data$Publication.Year

> str(work.data)

'data.frame': 3702 obs. of 12 variables:

$ Publication.Year : num 5 2 2 3 4 13 17 4 4 2 ...

$ Count.of.Simple.Family.Members : int 15 13 11 12 15 28 20 8 9 11 ...

$ Simple.Family.Cited.by.Count : int 91 15 32 63 16 95 64 59 62 168 ...

$ Count.of.Other.References : int 0 0 2 3 0 0 14 7 0 0 ...

$ Count.of.Cited.by.Patents : int 0 0 0 1 0 0 20 9 2 0 ...

$ Count.of.Cites.Patents : int 0 21 14 29 4 3 41 23 5 0 ...

$ Count.of.Cited.by.Patents.Within.3.years: int 0 0 0 0 0 0 4 9 2 0 ...

$ Count.of.Cited.by.Patents.Within.5.years: int 0 0 0 0 0 0 7 9 2 0 ...

$ Simple.Legal.Status : Factor w/ 5 levels "","Active","Inactive",..: 2 2 2 2 2 2 2 2 2 4 ...

$ Count.of.claims : int 40 57 20 18 15 17 39 22 13 16 ...

$ Quality.of.Family : Factor w/ 2 levels "High","Low": 1 1 1 1 1 1 1 1 1 1 ...

$ Classifier : Factor w/ 2 levels "Strong","Weak": 1 1 1 1 1 1 1 1 1 1 ...

> describe(work.data)

vars n mean sd median trimmed mad min max range skew

Publication.Year 1 3702 35.19 35.70 18 30.33 22.24 1 117 116 0.98

Count.of.Simple.Family.Members 2 3702 2.36 3.32 1 1.51 0.00 1 44 43 4.36

Simple.Family.Cited.by.Count 3 3702 12.07 31.21 1 4.87 1.48 0 585 585 6.68

Count.of.Other.References 4 3702 0.33 1.95 0 0.03 0.00 0 67 67 17.42

Count.of.Cited.by.Patents 5 3702 7.57 20.91 0 2.58 0.00 0 456 456 6.96

Count.of.Cites.Patents 6 3702 4.92 44.04 0 2.21 0.00 0 2624 2624 56.88

Count.of.Cited.by.Patents.Within.3.years 7 3702 1.17 3.86 0 0.31 0.00 0 70 70 7.52

Count.of.Cited.by.Patents.Within.5.years 8 3702 1.95 6.47 0 0.54 0.00 0 148 148 8.68

Simple.Legal.Status\* 9 3702 3.04 0.69 3 2.97 0.00 1 5 4 1.09

Count.of.claims 10 3702 17.12 22.60 10 13.06 11.86 1 402 401 5.65

Quality.of.Family\* 11 3702 1.71 0.45 2 1.76 0.00 1 2 1 -0.93

Classifier\* 12 3702 1.29 0.45 1 1.23 0.00 1 2 1 0.94

kurtosis se

Publication.Year -0.48 0.59

Count.of.Simple.Family.Members 27.01 0.05

Simple.Family.Cited.by.Count 73.64 0.51

Count.of.Other.References 456.23 0.03

Count.of.Cited.by.Patents 88.50 0.34

Count.of.Cites.Patents 3377.17 0.72

Count.of.Cited.by.Patents.Within.3.years 83.33 0.06

Count.of.Cited.by.Patents.Within.5.years 123.65 0.11

Simple.Legal.Status\* 2.47 0.01

Count.of.claims 62.86 0.37

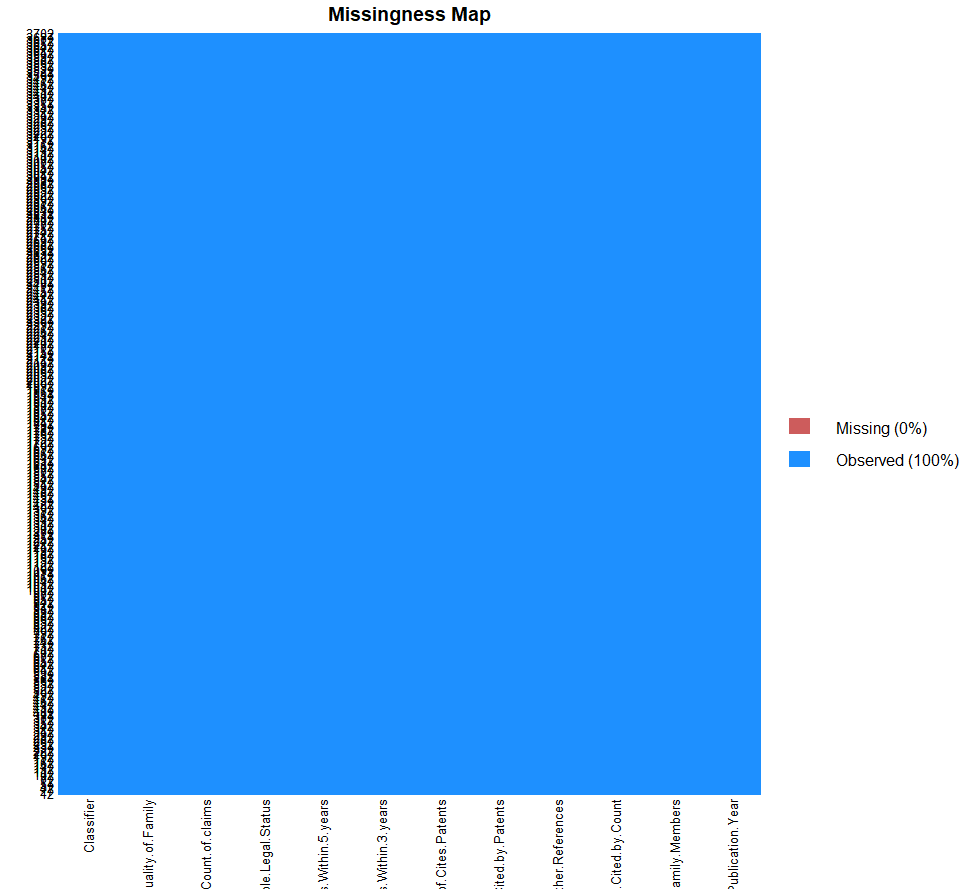
Quality.of.Family\* -1.13 0.01

Classifier\* -1.12 0.01

>

> #visualize the missing data

> missmap(work.data)



# Naïve Bayes

> #Data Visualization

>

> #Visual 1

> ggplot(work.data, aes(Publication.Year, colour = Classifier))+

+ geom\_freqpoly(binwidth = 1) + labs(title = "Publication Year Distribution by Classifier")

>

> #Visual 2

> c <- ggplot(work.data, aes(x=Count.of.Simple.Family.Members, fill=Classifier, color=Classifier)) +

+ geom\_histogram(binwidth = 1) + labs(title="Count.of.Simple.Family.Members Distribution by Classifier")

> c + theme\_bw()

>

#Building a model

> #split data into training and test data sets

> indxTrain <- createDataPartition(y = work.data$Classifier, p = 0.7, list = F)

> training <- work.data[indxTrain, ]

> testing <- work.data[-indxTrain, ]

>

> #Check dimensions of the split

> prop.table(table(work.data$Classifier))\*100

Strong Weak

71.25878 28.74122

>

> prop.table(table(training$Classifier))\*100

Strong Weak

71.25772 28.74228

> prop.table(table(testing$Classifier))\*100

Strong Weak

71.26126 28.73874

>

> #create objects x which holds the predictor variables and y which holds the response variables

> x = training[,-12]

> y = training$Classifier

>

> library(e1071)

Warning message:

пакет ‘e1071’ был собран под R версии 3.5.3

> model = train(x,y,'nb',trControl=trainControl(method='cv',number=10))

There were 50 or more warnings (use warnings() to see the first 50)

> model

Naive Bayes

2592 samples

11 predictor

2 classes: 'Strong', 'Weak'

No pre-processing

Resampling: Cross-Validated (10 fold)

Summary of sample sizes: 2333, 2332, 2332, 2332, 2333, 2333, ...

Resampling results across tuning parameters:

usekernel Accuracy Kappa

FALSE 0.5389279 0.2191585

TRUE 0.8209707 0.6032218

Tuning parameter 'fL' was held constant at a value of 0

Tuning parameter 'adjust' was held constant at a

value of 1

Accuracy was used to select the optimal model using the largest value.

The final values used for the model were fL = 0, usekernel = TRUE and adjust = 1.

>

> #Model Evaluation

> #Predict testing set

> Predict <- predict(model, newdata = testing)

There were 50 or more warnings (use warnings() to see the first 50)

>

> #Get the confusion matrix to see accuracy value and other parameter values

> confusionMatrix(Predict, testing$Classifier)

Confusion Matrix and Statistics

Reference

Prediction Strong Weak

Strong 643 36

Weak 148 283

Accuracy : 0.8342

95% CI : (0.811, 0.8557)

No Information Rate : 0.7126

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.6337

Mcnemar's Test P-Value : 2.768e-16

Sensitivity : 0.8129

Specificity : 0.8871

Pos Pred Value : 0.9470

Neg Pred Value : 0.6566

Prevalence : 0.7126

Detection Rate : 0.5793

Detection Prevalence : 0.6117

Balanced Accuracy : 0.8500

'Positive' Class : Strong

>

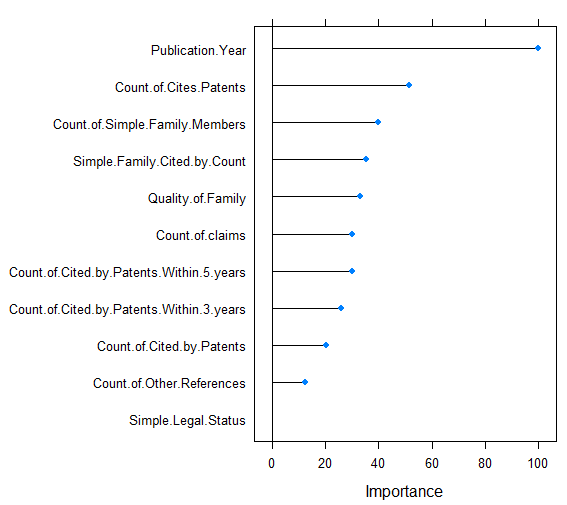
>

#Plot Variable performance

> X <- varImp(model)

> plot(X)

>



# Reduction the number of variables ---------------------------------------

>

> red.work.data <- data[c('Publication.Year', 'Count.of.Simple.Family.Members', 'Count.of.Cites.Patents',

+ 'Simple.Legal.Status', 'Count.of.claims', 'Quality.of.Family', 'Classifier')]

>

> #Building a model

> #split data into training and test data sets

> red.indxTrain <- createDataPartition(y = red.work.data$Classifier, p = 0.7, list = F)

> red.training <- red.work.data[indxTrain, ]

> red.testing <- red.work.data[-indxTrain, ]

>

> #Check dimensions of the split

> prop.table(table(red.work.data$Classifier))\*100

Strong Weak

71.25878 28.74122

>

> prop.table(table(red.training$Classifier))\*100

Strong Weak

71.25772 28.74228

> prop.table(table(red.testing$Classifier))\*100

Strong Weak

71.26126 28.73874

>

> #create objects x which holds the predictor variables and y which holds the response variables

> red.x = red.training[,-7]

> red.y = red.training$Classifier

>

> library(e1071)

> red.model = train(red.x,red.y,'nb',trControl=trainControl(method='LOOCV',number=10))

There were 50 or more warnings (use warnings() to see the first 50)

> red.model

Naive Bayes

2592 samples

6 predictor

2 classes: 'Strong', 'Weak'

No pre-processing

Resampling: Leave-One-Out Cross-Validation

Summary of sample sizes: 2591, 2591, 2591, 2591, 2591, 2591, ...

Resampling results across tuning parameters:

usekernel Accuracy Kappa

FALSE 0.6847994 0.3961436

TRUE 0.8746142 0.7057973

Tuning parameter 'fL' was held constant at a value of 0

Tuning parameter 'adjust' was held constant at a

value of 1

Accuracy was used to select the optimal model using the largest value.

The final values used for the model were fL = 0, usekernel = TRUE and adjust = 1.

>

> #Model Evaluation

> #Predict testing set

> red.Predict <- predict(red.model, newdata = red.testing)

There were 50 or more warnings (use warnings() to see the first 50)

>

> #Get the confusion matrix to see accuracy value and other parameter values

> confusionMatrix(red.Predict, red.testing$Classifier)

Confusion Matrix and Statistics

Reference

Prediction Strong Weak

Strong 702 42

Weak 89 277

Accuracy : 0.882

95% CI : (0.8615, 0.9004)

No Information Rate : 0.7126

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.724

Mcnemar's Test P-Value : 5.844e-05

Sensitivity : 0.8875

Specificity : 0.8683

Pos Pred Value : 0.9435

Neg Pred Value : 0.7568

Prevalence : 0.7126

Detection Rate : 0.6324

Detection Prevalence : 0.6703

Balanced Accuracy : 0.8779

'Positive' Class : Strong

# Random Forest

library(randomForest)

>

> #Building a model

> #split data into training and test data sets

> rf.indxTrain <- createDataPartition(y = work.data$Classifier, p = 0.7, list = F)

> rf.training <- work.data[rf.indxTrain, ]

> rf.testing <- work.data[-rf.indxTrain, ]

>

> rf.testing.unlabeled <- rf.testing[, -12]

> rf.testing.labels <- rf.testing$Classifier

>

> set.seed(123)

> # Training using &lsquo;random forest&rsquo; algorithm

> rf.model <- train(Classifier ~ Publication.Year + Count.of.Simple.Family.Members + Simple.Family.Cited.by.Count +

+ Count.of.Other.References + Count.of.Cited.by.Patents + Count.of.Cites.Patents +

+ Simple.Legal.Status + Count.of.claims + Quality.of.Family, # Survived is a function of the variables we decided to include

+ data = rf.training, # Use the train data frame as the training data

+ method = 'rf',# Use the 'random forest' algorithm

+ trControl = trainControl(method = 'cv', # Use cross-validation

+ number = 5)) # Use 5 folds for cross-validation

>

> rf.testing.unlabeled$Pred.Classifier <- predict(model, newdata = rf.testing)

There were 50 or more warnings (use warnings() to see the first 50)

> confusionMatrix(rf.testing.unlabeled$Pred.Classifier, rf.testing$Classifier)

Confusion Matrix and Statistics

Reference

Prediction Strong Weak

Strong 548 50

Weak 243 269

Accuracy : 0.736

95% CI : (0.7091, 0.7618)

No Information Rate : 0.7126

P-Value [Acc > NIR] : 0.04452

Kappa : 0.4541

Mcnemar's Test P-Value : < 2e-16

Sensitivity : 0.6928

Specificity : 0.8433

Pos Pred Value : 0.9164

Neg Pred Value : 0.5254

Prevalence : 0.7126

Detection Rate : 0.4937

Detection Prevalence : 0.5387

Balanced Accuracy : 0.7680

'Positive' Class : Strong

# K-Nearest Neighbors

#Normalization

> normalize <- function(x) {

+ return ((x - min(x)) / (max(x) - min(x))) }

>

> knn.work.data <- work.data

> knn.work.data$Simple.Legal.Status <- as.character(knn.work.data$Simple.Legal.Status)

> knn.work.data$Simple.Legal.Status[knn.work.data$Simple.Legal.Status == "Active"] <- 4

> knn.work.data$Simple.Legal.Status[knn.work.data$Simple.Legal.Status == "Inactive"] <- 3

> knn.work.data$Simple.Legal.Status[knn.work.data$Simple.Legal.Status == "Pending"] <- 2

> knn.work.data$Simple.Legal.Status[knn.work.data$Simple.Legal.Status == "Undetermined"] <- 1

> knn.work.data$Simple.Legal.Status <- as.numeric(knn.work.data$Simple.Legal.Status)

>

> knn.work.data$Quality.of.Family <- as.character(knn.work.data$Quality.of.Family)

> knn.work.data$Quality.of.Family[knn.work.data$Quality.of.Family == "High"] <- 1

> knn.work.data$Quality.of.Family[knn.work.data$Quality.of.Family == "Low"] <- 0

> knn.work.data$Quality.of.Family <- as.numeric(knn.work.data$Quality.of.Family)

>

> knn.work.data$Classifier <- as.character(knn.work.data$Classifier)

> knn.work.data$Classifier[knn.work.data$Classifier == "Strong"] <- 1

> knn.work.data$Classifier[knn.work.data$Classifier == "Weak"] <- 0

> knn.work.data$Classifier <- as.numeric(knn.work.data$Classifier)

>

> knn.work.data.n <- as.data.frame(lapply(knn.work.data[,1:11], normalize))

> knn.work.data.n$Simple.Legal.Status <- NULL

>

> #Building a model

> #split data into training and test data sets

> knn.indxTrain <- createDataPartition(y = work.data$Classifier, p = 0.7, list = F)

> knn.training <- knn.work.data.n[knn.indxTrain, ]

> knn.testing <- knn.work.data.n[-knn.indxTrain, ]

>

> #Creating seperate dataframe for 'Classifier' feature which is our target.

> train.knn.labels <- knn.work.data[knn.indxTrain,12]

> test.knn.labels <-knn.work.data[-knn.indxTrain,12]

>

> library(class)

> sqrt(NROW(train.knn.labels))

[1] 50.91169

> #We have 2593 observations in our training data set.

> #The square root of 2593 is around 50.92, therefore we’ll create two models.

> #One with ‘K’ value as 50 and the other model with a ‘K’ value as 51.

> knn.50 <- knn(train=knn.training, test=knn.testing, cl=train.knn.labels, k=50)

> knn.51 <- knn(train=knn.training, test=knn.testing, cl=train.knn.labels, k=51)

>

> #Model Evaluation

> #Calculate the proportion of correct classification for k = 26, 27

> ACC.50 <- 100 \* sum(test.knn.labels == knn.50)/NROW(test.knn.labels)

> ACC.51 <- 100 \* sum(test.knn.labels == knn.51)/NROW(test.knn.labels)

>

> # Check prediction against actual value in tabular form for k=50

> table(knn.50 ,test.knn.labels)

test.knn.labels

knn.50 0 1

0 268 25

1 51 766

> # Check prediction against actual value in tabular form for k=51

> table(knn.51 ,test.knn.labels)

test.knn.labels

knn.51 0 1

0 268 25

1 51 766

>

> confusionMatrix(table(knn.50, test.knn.labels))

Confusion Matrix and Statistics

test.knn.labels

knn.50 0 1

0 268 25

1 51 766

Accuracy : 0.9315

95% CI : (0.915, 0.9457)

No Information Rate : 0.7126

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8287

Mcnemar's Test P-Value : 0.004135

Sensitivity : 0.8401

Specificity : 0.9684

Pos Pred Value : 0.9147

Neg Pred Value : 0.9376

Prevalence : 0.2874

Detection Rate : 0.2414

Detection Prevalence : 0.2640

Balanced Accuracy : 0.9043

'Positive' Class : 0

> confusionMatrix(table(knn.51, test.knn.labels))

Confusion Matrix and Statistics

test.knn.labels

knn.51 0 1

0 268 25

1 51 766

Accuracy : 0.9315

95% CI : (0.915, 0.9457)

No Information Rate : 0.7126

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8287

Mcnemar's Test P-Value : 0.004135

Sensitivity : 0.8401

Specificity : 0.9684

Pos Pred Value : 0.9147

Neg Pred Value : 0.9376

Prevalence : 0.2874

Detection Rate : 0.2414

Detection Prevalence : 0.2640

Balanced Accuracy : 0.9043

'Positive' Class : 0

>

> #Optimization

> i=1

> k.optm=1

> for (i in 1:51){

+ knn.mod <- knn(train=knn.training, test=knn.testing, cl=train.knn.labels, k=i)

+ k.optm[i] <- 100 \* sum(test.knn.labels == knn.mod)/NROW(test.knn.labels)

+ k=i

+ cat(k,'=',k.optm[i],'

+ ')

+ }

1 = 86.75676

2 = 88.1982

3 = 91.62162

4 = 91.89189

5 = 92.79279

6 = 92.88288

7 = 93.15315

8 = 92.97297

9 = 93.06306

10 = 93.15315

11 = 93.06306

12 = 93.15315

13 = 93.06306

14 = 93.06306

15 = 93.06306

16 = 93.06306

17 = 93.06306

18 = 93.06306

19 = 93.06306

20 = 92.97297

21 = 93.06306

22 = 93.06306

23 = 93.06306

24 = 93.06306

25 = 93.06306

26 = 93.15315

27 = 93.15315

28 = 93.15315

29 = 93.15315

30 = 93.15315

31 = 93.15315

32 = 93.15315

33 = 93.15315

34 = 93.15315

35 = 93.15315

36 = 93.15315

37 = 93.15315

38 = 93.15315

39 = 93.15315

40 = 93.15315

41 = 93.15315

42 = 93.24324

43 = 93.24324

44 = 93.24324

45 = 93.24324

46 = 93.24324

47 = 93.15315

48 = 93.15315

49 = 93.15315

50 = 93.15315

51 = 93.15315

#Correction

> knn.42 <- knn(train=knn.training, test=knn.testing, cl=train.knn.labels, k=42)

> ACC.42 <- 100 \* sum(test.knn.labels == knn.42)/NROW(test.knn.labels)

> ACC.42

[1] 93.24324

> # Check prediction against actual value in tabular form for k=50

> table(knn.42 ,test.knn.labels)

test.knn.labels

knn.42 0 1

0 269 25

1 50 766

> confusionMatrix(table(knn.42, test.knn.labels))

Confusion Matrix and Statistics

test.knn.labels

knn.42 0 1

0 269 25

1 50 766

Accuracy : 0.9324

95% CI : (0.916, 0.9465)

No Information Rate : 0.7126

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8311

Mcnemar's Test P-Value : 0.005584

Sensitivity : 0.8433

Specificity : 0.9684

Pos Pred Value : 0.9150

Neg Pred Value : 0.9387

Prevalence : 0.2874

Detection Rate : 0.2423

Detection Prevalence : 0.2649

Balanced Accuracy : 0.9058

'Positive' Class : 0

# Support-vector machines

library(caret)

>

> svm.work.data <- work.data

> svm.work.data$Simple.Legal.Status <- as.character(svm.work.data$Simple.Legal.Status)

> svm.work.data$Simple.Legal.Status[svm.work.data$Simple.Legal.Status == "Active"] <- 4

> svm.work.data$Simple.Legal.Status[svm.work.data$Simple.Legal.Status == "Inactive"] <- 3

> svm.work.data$Simple.Legal.Status[svm.work.data$Simple.Legal.Status == "Pending"] <- 2

> svm.work.data$Simple.Legal.Status[svm.work.data$Simple.Legal.Status == "Undetermined"] <- 1

> svm.work.data$Simple.Legal.Status <- as.numeric(svm.work.data$Simple.Legal.Status)

>

> svm.work.data$Quality.of.Family <- as.character(svm.work.data$Quality.of.Family)

> svm.work.data$Quality.of.Family[svm.work.data$Quality.of.Family == "High"] <- 1

> svm.work.data$Quality.of.Family[svm.work.data$Quality.of.Family == "Low"] <- 0

> svm.work.data$Quality.of.Family <- as.numeric(svm.work.data$Quality.of.Family)

>

> svm.work.data$Classifier <- as.character(svm.work.data$Classifier)

> svm.work.data$Classifier[svm.work.data$Classifier == "Strong"] <- 1

> svm.work.data$Classifier[svm.work.data$Classifier == "Weak"] <- 0

> svm.work.data$Classifier <- as.numeric(svm.work.data$Classifier)

>

> svm.work.data.n <- as.data.frame(lapply(svm.work.data[,1:11], normalize))

> svm.work.data.n$Simple.Legal.Status <- NULL

> svm.work.data[is.na(svm.work.data)] <- 0

> #Building a model

> #split data into training and test data sets

> svm.indxTrain <- createDataPartition(y = svm.work.data$Classifier, p = 0.7, list = F)

> svm.training <- svm.work.data[svm.indxTrain, ]

> svm.testing <- svm.work.data[-svm.indxTrain, ]

>

> svm.training[['Classifier']] = factor(svm.training[["Classifier"]])

> #Training a model

> svm.traincontrol <- trainControl(method = 'repeatedcv', number = 10, repeats = 4)

> svm\_linear <- train(Classifier ~., data = svm.training, method = 'svmLinear', trControl = svm.traincontrol,

+ preProcess = c('center', 'scale'), tuneLength = 10)

> svm\_linear

Support Vector Machines with Linear Kernel

2592 samples

11 predictor

2 classes: '0', '1'

Pre-processing: centered (11), scaled (11)

Resampling: Cross-Validated (10 fold, repeated 4 times)

Summary of sample sizes: 2332, 2333, 2333, 2333, 2333, 2333, ...

Resampling results:

Accuracy Kappa

0.914455 0.7859152

Tuning parameter 'C' was held constant at a value of 1

>

> svm.test.predict <- predict(svm\_linear, newdata = svm.testing)

> confusionMatrix(table(svm.test.predict, svm.testing$Classifier))

Confusion Matrix and Statistics

svm.test.predict 0 1

0 269 23

1 44 774

Accuracy : 0.9396

95% CI : (0.924, 0.9529)

No Information Rate : 0.718

P-Value [Acc > NIR] : < 2e-16

Kappa : 0.8478

Mcnemar's Test P-Value : 0.01455

Sensitivity : 0.8594

Specificity : 0.9711

Pos Pred Value : 0.9212

Neg Pred Value : 0.9462

Prevalence : 0.2820

Detection Rate : 0.2423

Detection Prevalence : 0.2631

Balanced Accuracy : 0.9153

'Positive' Class : 0

>

> #Selecting C-value in Linear classifier

> grid <- expand.grid(C = c(0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2,5))

> svm\_Linear\_Grid <- train(Classifier ~., data = svm.training, method = "svmLinear",

+ trControl=svm.traincontrol,

+ preProcess = c("center", "scale"),

+ tuneGrid = grid,

+ tuneLength = 10)

There were 42 warnings (use warnings() to see them)

> svm\_Linear\_Grid

Support Vector Machines with Linear Kernel

2592 samples

11 predictor

2 classes: '0', '1'

Pre-processing: centered (11), scaled (11)

Resampling: Cross-Validated (10 fold, repeated 4 times)

Summary of sample sizes: 2332, 2333, 2333, 2333, 2333, 2333, ...

Resampling results across tuning parameters:

C Accuracy Kappa

0.00 NaN NaN

0.01 0.9125178 0.7800108

0.05 0.9148341 0.7867097

0.10 0.9149302 0.7869775

0.25 0.9150267 0.7872492

0.50 0.9150267 0.7872871

0.75 0.9149302 0.7870240

1.00 0.9149302 0.7870240

1.25 0.9149302 0.7870240

1.50 0.9150267 0.7872954

1.75 0.9150267 0.7872954

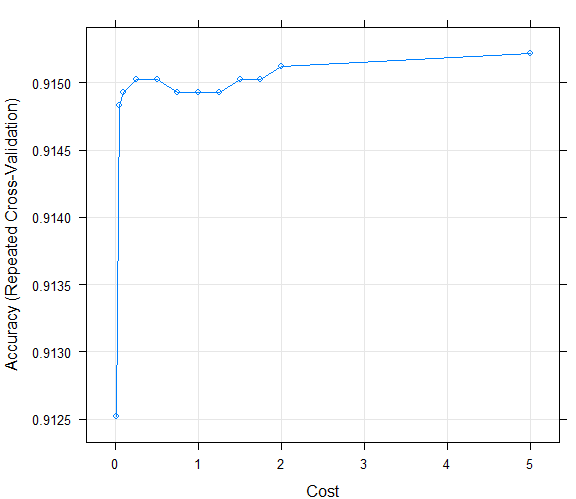
2.00 0.9151233 0.7875585

5.00 0.9152198 0.7878300

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was C = 5.

> plot(svm\_Linear\_Grid)



> #The plot is showing that our classifier is giving best accuracy on C = 0.05.

> test\_pred\_grid <- predict(svm\_Linear\_Grid, newdata = svm.testing)

> test\_pred\_grid

[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

[54] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

[107] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

[160] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

[213] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1

[266] 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 0 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0

[319] 1 0 0 0 0 1 1 1 1 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0

[372] 0 1 0 0 1 1 1 0 1 1 0 0 0 0 1 1 0 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 1 0

[425] 1 0 1 1 1 1 1 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 1 1 1

[478] 0 1 1 1 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 1 1 0

[531] 1 1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 1 0 0 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0 1 1

[584] 1 1 1 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 0 1 0 0 1 1 1 1 1 0 0 1 1

[637] 1 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 1 1 0

[690] 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 0 1 1 1 1 1 1 0

[743] 1 1 1 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1

[796] 1 1 0 1 1 1 1 1 0 1 0 0 0 0 1 1 1 0 0 0 1 1 0 0 1 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1

[849] 1 1 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 1 0 1 1 1 1 1 0 1 1 0

[902] 1 1 1 1 0 0 1 1 1 0 1 1 0 1 0 0 0 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1

[955] 1 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 0 0 0 1

[ reached getOption("max.print") -- omitted 110 entries ]

Levels: 0 1

> confusionMatrix(table(test\_pred\_grid, svm.testing$Classifier))

Confusion Matrix and Statistics

test\_pred\_grid 0 1

0 269 23

1 44 774

Accuracy : 0.9396

95% CI : (0.924, 0.9529)

No Information Rate : 0.718

P-Value [Acc > NIR] : < 2e-16

Kappa : 0.8478

Mcnemar's Test P-Value : 0.01455

Sensitivity : 0.8594

Specificity : 0.9711

Pos Pred Value : 0.9212

Neg Pred Value : 0.9462

Prevalence : 0.2820

Detection Rate : 0.2423

Detection Prevalence : 0.2631

Balanced Accuracy : 0.9153

'Positive' Class : 0