Classification Techniques Assignment–Audit

Saixiong Han (sah178)

Feburary 12th, 2017

Task: Analyze dataset analyze dataset audit.csv. The objective is to predict the binary (TARGET_Adjusted) target variable.

Requirement: Apply different classification techniques (incl. logistic regression, kNN, Naive Bayesian, decision tree, SVM, and Ensemble methods) on this dataset. Use all available predictors in your models.

1. Use a 10-fold cross-validation to evaluate different classification techniques

Requirement: Report your 10-fold CV classification results in a performance table. In the table, report the values of different performance measures for each classification technique.

a. Data Preprocess

Import the data into R and get the summary from the original data.

```
audit <- read.csv("C:/Users/daisy/OneDrive/Study/DM/week3/audit.csv", header = TRUE,
    sep = ",", stringsAsFactors = TRUE)
head(audit)
```

```
##        ID Age Employment Education   Marital Occupation    Income Gender
## 1 1004641  38    Private   College Unmarried    Service  81838.00 Female
## 2 1010229  35    Private Associate    Absent  Transport  72099.00   Male
## 3 1024587  32    Private    HSgrad  Divorced   Clerical 154676.74   Male
## 4 1038288  45    Private  Bachelor   Married     Repair  27743.82   Male
## 5 1044221  60    Private   College   Married  Executive   7568.23   Male
## 6 1047095  74    Private    HSgrad   Married    Service  33144.40   Male
##   Deductions Hours RISK_Adjustment TARGET_Adjusted
## 1          0    72               0               0
## 2          0    30               0               0
## 3          0    40               0               0
## 4          0    55            7298               1
## 5          0    40           15024               1
## 6          0    30               0               0
```

From the head we can see, there are two columns which are useless in the analysis. One is ID and the other is RISK_Adjustment. We can drop these two columns first and take the rest columns into model.

```
audit1 <- audit[, 2:12]
audit1 <- audit1[, -10]
dim(audit1)
```

```
## [1] 2000   10
```

```
summary(audit1)
```

```
##       Age             Employment        Education
##  Min.   :17.00   Private    :1411   HSgrad    :660
##  1st Qu.:28.00   Consultant: 148   College   :442
##  Median :37.00   PSLocal    : 119   Bachelor  :345
```

```
##  Mean   :38.62   SelfEmp   :  79   Master    :102  
##  3rd Qu.:48.00   PSState   :  72   Vocational: 86  
##  Max.   :90.00   (Other)   :  71   Yr11      : 74  
##                  NA's      : 100   (Other)   :291  
##                   Marital            Occupation      Income        
##  Absent               :669   Executive   :289   Min.   :   609.7  
##  Divorced             :266   Professional:247   1st Qu.: 34433.1  
##  Married              :917   Clerical    :232   Median : 59768.9  
##  Married-spouse-absent: 22   Repair      :225   Mean   : 84688.5  
##  Unmarried            : 67   Service     :210   3rd Qu.:113842.9  
##  Widowed              : 59   (Other)     :696   Max.   :481259.5  
##                              NA's        :101                     
##     Gender       Deductions          Hours       TARGET_Adjusted 
##  Female: 632   Min.   :   0.00   Min.   : 1.00   Min.   :0.0000  
##  Male  :1368   1st Qu.:   0.00   1st Qu.:38.00   1st Qu.:0.0000  
##                Median :   0.00   Median :40.00   Median :0.0000  
##                Mean   :  67.57   Mean   :40.07   Mean   :0.2315  
##                3rd Qu.:   0.00   3rd Qu.:45.00   3rd Qu.:0.0000  
##                Max.   :2904.00   Max.   :99.00   Max.   :1.0000  
## 
```

From the summary of the new data we can see, there are 10 variables in the dataset. Except TARGET_Adjusted, the other variables are all predictors. There are some missing value in "Employment" and "Occupation", we can exclude these missing values and take the rest rows into model.

```
audit2 <- na.omit(audit1)
summary(audit2)
```

```
##       Age            Employment        Education  
##  Min.   :17.0   Private   :1411   HSgrad    :633  
##  1st Qu.:28.0   Consultant: 148   College   :418  
##  Median :37.0   PSLocal   : 119   Bachelor  :332  
##  Mean   :38.3   SelfEmp   :  79   Master    : 98  
##  3rd Qu.:47.0   PSState   :  72   Vocational: 81  
##  Max.   :83.0   PSFederal :  69   Associate : 67  
##                 (Other)   :   1   (Other)   :270  
##                   Marital            Occupation      Income        
##  Absent               :633   Executive   :289   Min.   :   609.7  
##  Divorced             :256   Professional:247   1st Qu.: 33987.2  
##  Married              :878   Clerical    :232   Median : 59534.9  
##  Married-spouse-absent: 21   Repair      :225   Mean   : 84404.9  
##  Unmarried            : 64   Service     :210   3rd Qu.:113331.2  
##  Widowed              : 47   Sales       :206   Max.   :481259.5  
##                              (Other)     :490                     
##     Gender       Deductions          Hours       TARGET_Adjusted 
##  Female: 592   Min.   :   0.00   Min.   : 1.00   Min.   :0.0000  
##  Male  :1307   1st Qu.:   0.00   1st Qu.:40.00   1st Qu.:0.0000  
##                Median :   0.00   Median :40.00   Median :0.0000  
##                Mean   :  68.66   Mean   :40.57   Mean   :0.2354  
##                3rd Qu.:   0.00   3rd Qu.:45.00   3rd Qu.:0.0000  
##                Max.   :2824.00   Max.   :99.00   Max.   :1.0000  
## 
```

```
audit2[1:3, ]
```

```
##   Age Employment Education   Marital Occupation   Income Gender Deductions
```

```
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours TARGET_Adjusted
## 1    72               0
## 2    30               0
## 3    40               0
```

The summary shows there is no missing value in the Employment and Occupation. While the first time I tried to take this dataset into model, I got some problems with variables "Employment" and "Occupation". I tried to check the levels in these two variables and found the problem.

```
audit3 = audit2
summary(audit3$Employment)
```

```
## Consultant    Private  PSFederal    PSLocal    PSState    SelfEmp
##        148       1411         69        119         72         79
## Unemployed  Volunteer
##          0          1
```

```
summary(audit3$Occupation)
```

```
##      Cleaner     Clerical    Executive      Farming         Home
##           91          232          289           58            5
##    Machinist     Military Professional   Protective       Repair
##          139            1          247           40          225
##        Sales      Service      Support    Transport
##          206          210           49          107
```

In Employment, there are 8 levels, while Volunteer only has one row and Unemployed has no row at all. We need to exclude these rows and levels from the dataset, so that the train model and test model can have the same levels. It is the same with the variable "Occupation". In Occupation, there are only one row whose level is Military. If we keep the row here, that means this row either in train dataset or in test dataset. And wherever it is, the train dataset and test dataset can't have the same level number. Thus, the model can't be run successfully. We need to delete the row and level Military from Occupation too.

```
audit3 = audit2
audit3 = subset(audit3, !(Employment == "Volunteer"))
audit3 = subset(audit3, !(Employment == "Unemployed"))
audit3 = subset(audit3, !(Occupation == "Military"))

levels(droplevels(audit3$Employment))
```

```
## [1] "Consultant" "Private"    "PSFederal"  "PSLocal"    "PSState"
## [6] "SelfEmp"
```

```
levels(droplevels(audit3$Occupation))
```

```
##  [1] "Cleaner"      "Clerical"     "Executive"    "Farming"
##  [5] "Home"         "Machinist"    "Professional" "Protective"
##  [9] "Repair"       "Sales"        "Service"      "Support"
## [13] "Transport"
```

Now our dataset can be applied to the model. We just need to change the target variables to continues variable y. After that, we can use the 10-fould cross valiadation on the different classification techniques with the audit dataset.

```
audit3$y = as.numeric(audit3$TARGET_Adjusted)
audit3 = audit3[, -10]
```

Import the packages to R and set the seed.

```
library(MASS)
library(plyr)
library(ROCR)
library(e1071)
library(rpart)
library(ada)
library(class)

set.seed(1)
```

b. Function Definetion

Define the function of my.classification which includes pre.test function and cross valiadation function. We will use this function as the main function when we test the different classification models.

```
my.classifier <- function(dataset, cl.name = "knn", do.cv = F, n) {
    n.obs <- nrow(dataset)
    n.cols <- ncol(dataset)
    cat("my dataset:", n.obs, "observations", n.cols - 1, "predictors", "\n")
    print(dataset[1:3, ])
    cat("label (y) distribution:")
    print(table(dataset$y))

    pre.test(dataset, cl.name, n)
    if (do.cv)
        k.fold.cv(dataset, cl.name, n)
}
```

Define the k-fold cross valiadation function which can split the data into train dataset and test dataset. Meanwhile, it will do the cross valiadation on the train dataset and test dataset for 10 times and get the avarage error, precision, recall, f-score and AUC for each model by using performance. By caculating the probability and actual value, we can get confusion matrix and ROC plot for each model.

```
k.fold.cv <- function(dataset, cl.name, n, k.fold = 10, prob.cutoff = 0.5) {
    n.obs <- nrow(dataset)
    s = sample(n.obs)
    errors = dim(k.fold)
    probs = NULL
    actuals = NULL
    for (k in 1:k.fold) {
        test.idx = which(s%%k.fold == (k - 1))
        train.set = dataset[-test.idx, ]
        test.set = dataset[test.idx, ]
        cat(k.fold, "-fold CV run", k, cl.name, ":", "#training:", nrow(train.set),
            "#testing", nrow(test.set), "\n")
        prob = do.classification(train.set, test.set, cl.name, n)
        predicted = as.numeric(prob > prob.cutoff)
        actual = test.set$y
        confusion.matrix = table(actual, factor(predicted, levels = c(0, 1)))
        # confusion.matrix
        error = (confusion.matrix[1, 2] + confusion.matrix[2, 1])/nrow(test.set)
        # errors[k] = error
        cat("\t\terror=", error, "\n")
```

```
        probs = c(probs, prob)
        actuals = c(actuals, actual)
    }
    avg.error = mean(errors)
    cat(k.fold, "-fold CV results:", "avg error=", avg.error, "\n")

    ## plot ROC
    result = data.frame(probs, actuals)
    pred = prediction(result$probs, result$actuals)
    perf = performance(pred, "tpr", "fpr")
    plot(perf)

    ## get other measures by using 'performance'
    get.measure <- function(pred, measure.name = "auc") {
        perf = performance(pred, measure.name)
        m <- unlist(slot(perf, "y.values"))
        m
    }
    err = mean(get.measure(pred, "err"))
    precision = mean(get.measure(pred, "prec"), na.rm = T)
    recall = mean(get.measure(pred, "rec"), na.rm = T)
    fscore = mean(get.measure(pred, "f"), na.rm = T)
    cat("error=", err, "precision=", precision, "recall=", recall, "f-score",
        fscore, "\n")
    auc = get.measure(pred, "auc")
    cat("auc=", auc, "\n")
}
```

Define the pre.teste function to apply each model to test dataset and get the probability of each binary target variable. Set the prob.cutoff as 0.5 and get the confusion matrix for each model.

```
pre.test <- function(dataset, cl.name, n, r = 0.6, prob.cutoff = 0.5) {
    n.obs <- nrow(dataset)
    n.train = floor(n.obs * r)
    train.idx = sample(1:n.obs, n.train)
    train.idx
    train.set = dataset[train.idx, ]
    test.set = dataset[-train.idx, ]
    cat("pre-test", cl.name, ":", "#training:", nrow(train.set), "#testing",
        nrow(test.set), "\n")
    colnames(train.set)
    prob = do.classification(train.set, test.set, cl.name, n)
    # An array of probabilities for cases being positive
    length(prob)

    ## get confusion matrix
    predicted = as.numeric(prob > prob.cutoff)
    actual = test.set$y
    confusion.matrix = table(actual, factor(predicted, levels = c(0, 1)))
    error = (confusion.matrix[1, 2] + confusion.matrix[2, 1])/nrow(test.set)
    cat("error rate:", error, "\n")

    ## plot ROC
    result = data.frame(prob, actual)
```

```
    pred = prediction(result$prob, result$actual)
    perf = performance(pred, "tpr", "fpr")
    plot(perf)
    return(pred)
}
```

Define the do.classification function which includes different classification techniques. I tried to add defalut value in Knn method and it can run successfully. However, for the SVM and decision tree model, since the parameters we need to change are hard to convert to some default number, We may need to redefine this function when we try different variants.

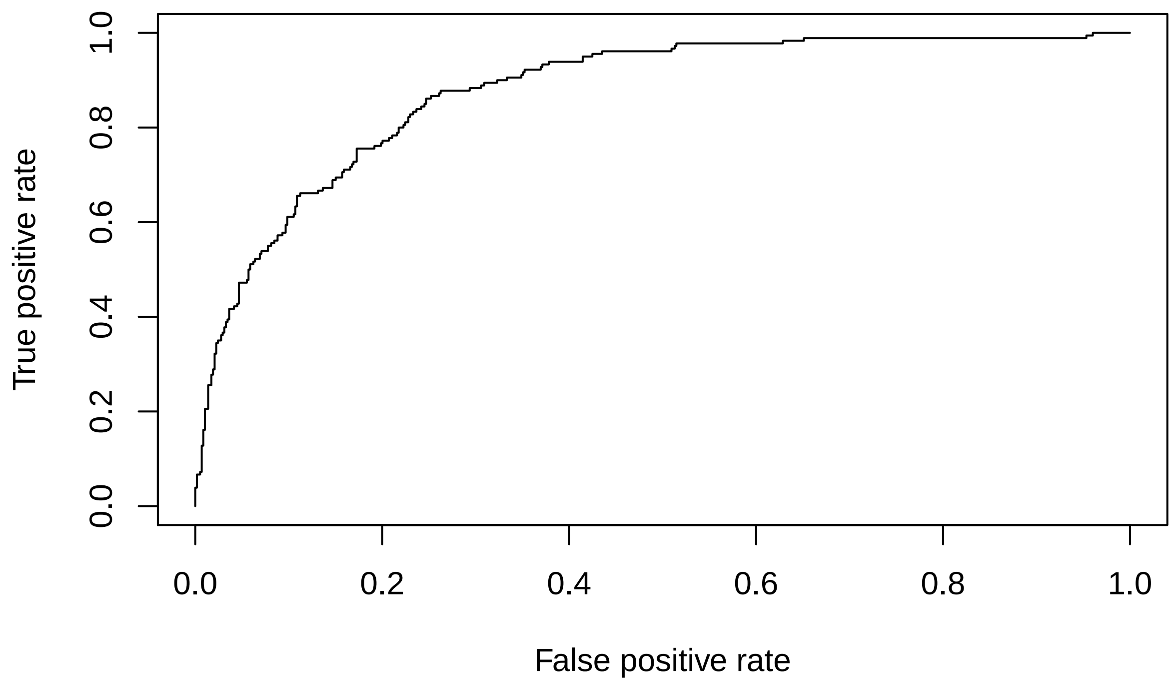
```
do.classification <- function(train.set, test.set, cl.name, n, verbose = F) {
    switch(cl.name, knn = {
        prob = knn(train.set[, -10], test.set[, -10], cl = train.set[, 10],
            k = n, prob = T)
        prob = attr(prob, "prob")
        attr(prob, "prob")[prob == 0] = 1 - attr(prob, "prob")[prob == 0]
        prob
    }, lr = {
        # logistic regression
        names(train.set)
        model = glm(y ~ ., family = binomial, data = train.set)
        if (verbose) {
            print(summary(model))
        }
        prob = predict(model, newdata = test.set, type = "response")
        prob
    }, nb = {
        # Naive Bayesian
        model = naiveBayes(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "raw")
        prob = prob[, 2]/rowSums(prob)  # renormalize the prob.
        prob
    }, dtree = {
        # Decision Tree
        model = rpart(y ~ ., data = train.set)
        test.set = test.set[, -10]
        if (verbose) {
            print(summary(model))
            printcp(model)
            plotcp(model)
            ## plot the tree
            plot(model, uniform = TRUE, main = "Classification Tree")
            text(model, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        prob = predict(model, newdata = test.set)

        if (0) {
            pfit <- prune(model, cp = model$cptable[which.min(model$cptable[,
                "xerror"]), "CP"])
            prob = predict(pfit, newdata = test.set)
            ## plot the pruned tree
            plot(pfit, uniform = TRUE, main = "Pruned Classification Tree")
            text(pfit, use.n = TRUE, all = TRUE, cex = 0.8)
```

```
        }
        head(prob)
        prob
    }, svm = {
        model = svm(y ~ ., data = train.set, probability = T)
        if (0) {
            tuned <- tune.svm(y ~ ., data = train.set, kernel = "radial", gamma = 10^(-6:-1),
                cost = 10^(-1:2))
            summary(tuned)
            gamma = tuned[["best.parameters"]]$gamma
            cost = tuned[["best.parameters"]]$cost
            model = svm(y ~ ., data = train.set, probability = T, kernel = "radial",
                gamma = gamma, cost = cost)
        }
        test.set = test.set[, -10]
        prob = predict(model, newdata = test.set, probability = T)
        dim(prob)
        prob
    }, ada = {
        model = ada(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "probs")
        prob = prob[, 2]/rowSums(prob)
        prob
    })
}
```

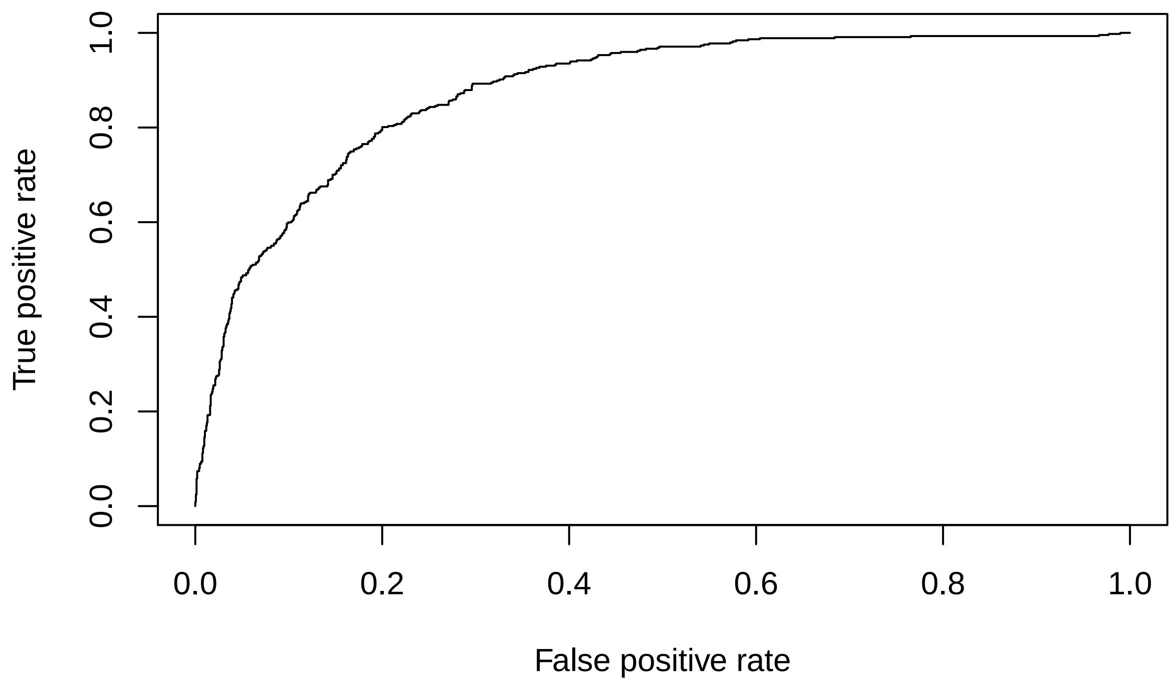
Logistic Regression Model

```
my.classifier(audit3, cl.name = "lr", do.cv = T)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education   Marital Occupation   Income Gender Deductions
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test lr : #training: 1138 #testing 759
## error rate: 0.1673254
```




```
## 10 -fold CV run 1 lr : #training: 1708 #testing 189
##      error= 0.1481481
## 10 -fold CV run 2 lr : #training: 1707 #testing 190
##      error= 0.1368421
## 10 -fold CV run 3 lr : #training: 1707 #testing 190
##      error= 0.1894737
## 10 -fold CV run 4 lr : #training: 1707 #testing 190
##      error= 0.1684211
## 10 -fold CV run 5 lr : #training: 1707 #testing 190
##      error= 0.1789474
## 10 -fold CV run 6 lr : #training: 1707 #testing 190
##      error= 0.2
## 10 -fold CV run 7 lr : #training: 1707 #testing 190
##      error= 0.2210526
## 10 -fold CV run 8 lr : #training: 1707 #testing 190
##      error= 0.1842105
## 10 -fold CV run 9 lr : #training: 1708 #testing 189
##      error= 0.1058201
## 10 -fold CV run 10 lr : #training: 1708 #testing 189
##      error= 0.1640212
## 10 -fold CV results: avg error= NA
```




```
## error= 0.3642535 precision= 0.4782371 recall= 0.7880437 f-score 0.5114609
## auc= 0.8770393
```

Knn model. Convert the categorical variables to numerical variables so that we can use Knn on the dataset.

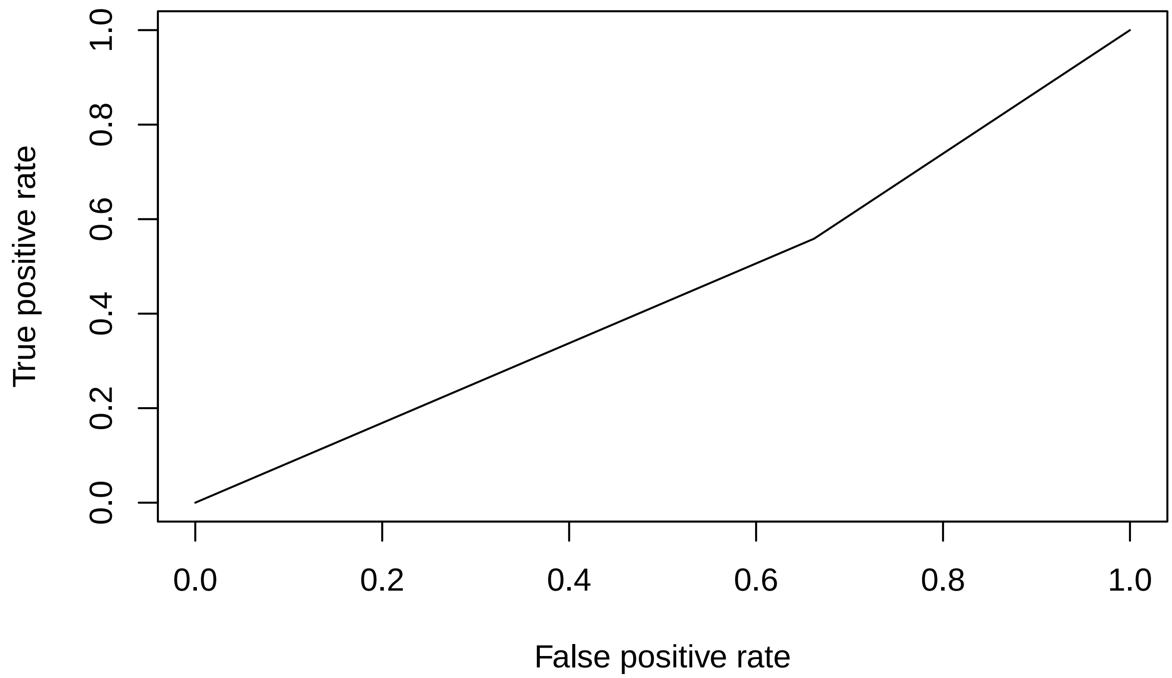
```
audit4 = audit3
audit4$Education = as.numeric(audit3$Education)
audit4$Employment = as.numeric(audit3$Employment)
audit4$Occupation = as.numeric(audit3$Occupation)
audit4$Age = as.numeric(audit3$Age)
audit4$Marital = as.numeric(audit3$Marital)
audit4$Gender = as.numeric(audit3$Gender)
audit4$Hours = as.numeric(audit3$Hours)
```

Try k=2

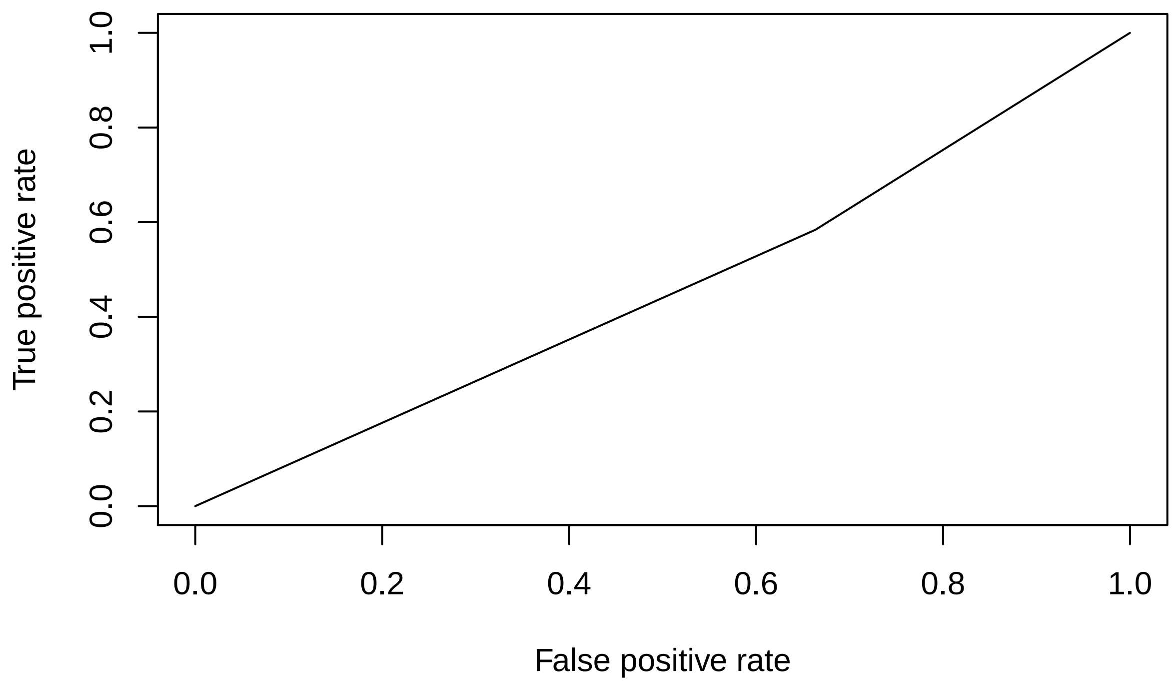
```
my.classifier(audit4, cl.name = "knn", do.cv = T, n = 2)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education Marital Occupation   Income Gender Deductions
## 1  38          2         3       5         12  81838.0      1          0
## 2  35          2         1       1         14  72099.0      2          0
## 3  32          2         5       2          2 154676.7      2          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
```

```
##    0    1
## 1450  447
## pre-test knn : #training: 1138 #testing 759
## error rate: 0.6100132
```




```
## 10 -fold CV run 1 knn : #training: 1708 #testing 189
##      error= 0.6296296
## 10 -fold CV run 2 knn : #training: 1707 #testing 190
##      error= 0.5684211
## 10 -fold CV run 3 knn : #training: 1707 #testing 190
##      error= 0.5947368
## 10 -fold CV run 4 knn : #training: 1707 #testing 190
##      error= 0.6473684
## 10 -fold CV run 5 knn : #training: 1707 #testing 190
##      error= 0.5789474
## 10 -fold CV run 6 knn : #training: 1707 #testing 190
##      error= 0.6210526
## 10 -fold CV run 7 knn : #training: 1707 #testing 190
##      error= 0.5789474
## 10 -fold CV run 8 knn : #training: 1707 #testing 190
##      error= 0.5947368
## 10 -fold CV run 9 knn : #training: 1708 #testing 189
##      error= 0.6084656
## 10 -fold CV run 10 knn : #training: 1708 #testing 189
##      error= 0.6296296
## 10 -fold CV results: avg error= NA
```



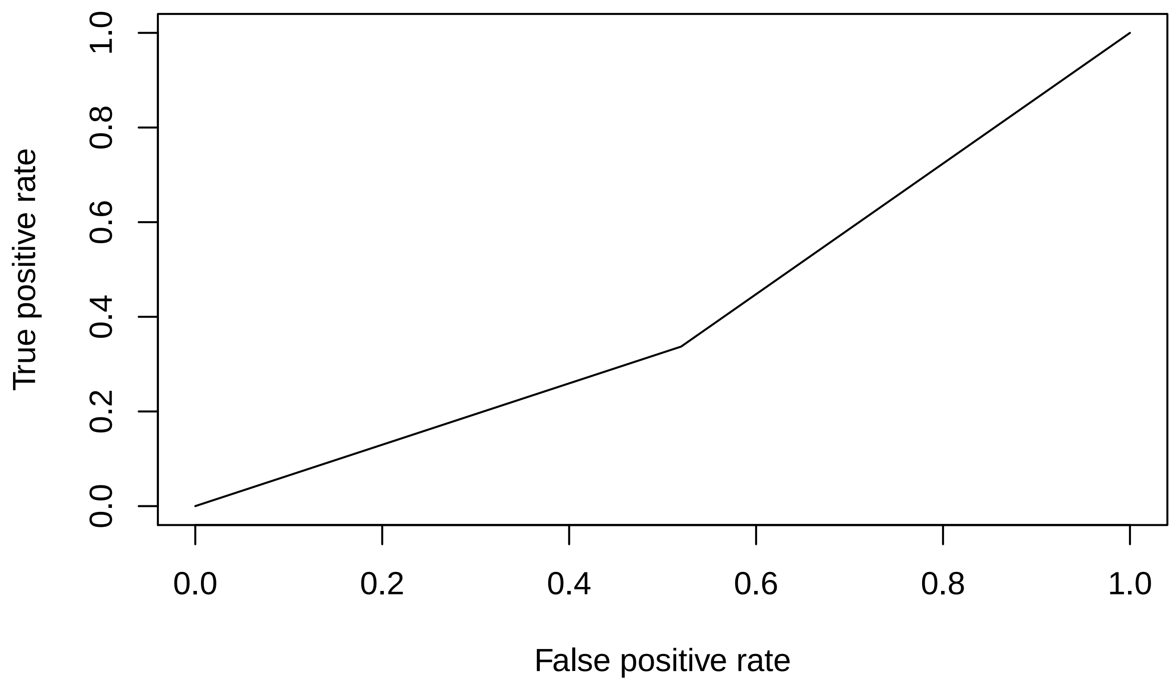

```
## error= 0.5350554 precision= 0.2245224 recall= 0.5279642 f-score 0.3469871
## auc= 0.4602222
```

2. Report at least two variants for techniques with parameters and incorporate them into your table.

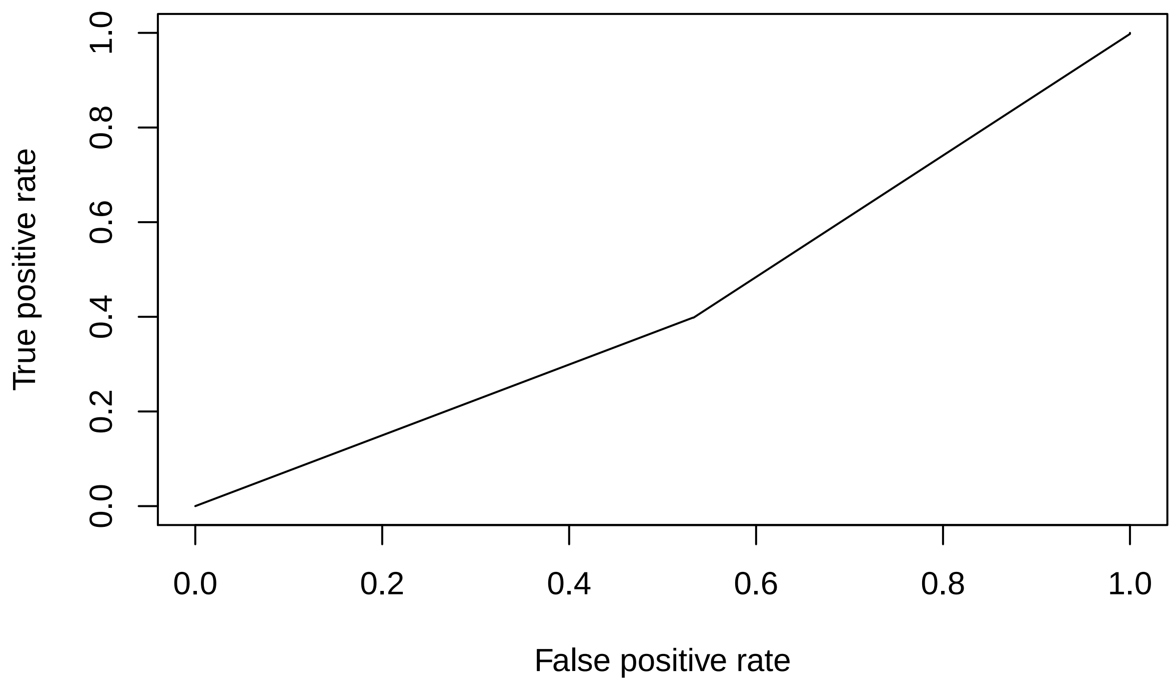
Try k=3

```
my.classifier(audit4, cl.name = "knn", do.cv = T, n = 3)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education Marital Occupation   Income Gender Deductions
## 1  38          2         3       5         12  81838.0      1          0
## 2  35          2         1       1         14  72099.0      2          0
## 3  32          2         5       2          2 154676.7      2          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test knn : #training: 1138 #testing 759
## error rate: 0.7654809
```




```
## 10 -fold CV run 1 knn : #training: 1708 #testing 189
##      error= 0.8042328
## 10 -fold CV run 2 knn : #training: 1707 #testing 190
##      error= 0.7421053
## 10 -fold CV run 3 knn : #training: 1707 #testing 190
##      error= 0.7105263
## 10 -fold CV run 4 knn : #training: 1707 #testing 190
##      error= 0.7578947
## 10 -fold CV run 5 knn : #training: 1707 #testing 190
##      error= 0.7894737
## 10 -fold CV run 6 knn : #training: 1707 #testing 190
##      error= 0.7684211
## 10 -fold CV run 7 knn : #training: 1707 #testing 190
##      error= 0.7842105
## 10 -fold CV run 8 knn : #training: 1707 #testing 190
##      error= 0.7578947
## 10 -fold CV run 9 knn : #training: 1708 #testing 189
##      error= 0.7989418
## 10 -fold CV run 10 knn : #training: 1708 #testing 189
##      error= 0.7354497
## 10 -fold CV results: avg error= NA
```

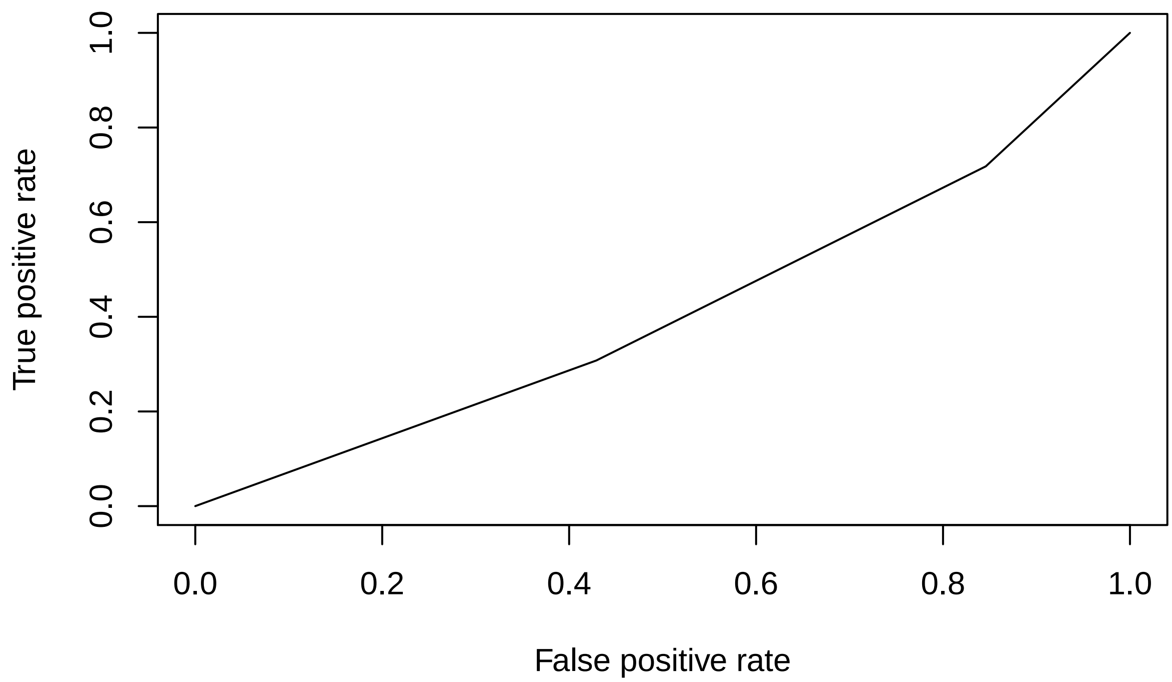



```
## error= 0.5725883 precision= 0.2113518 recall= 0.5588367 f-score 0.3178973
## auc= 0.4321685
```

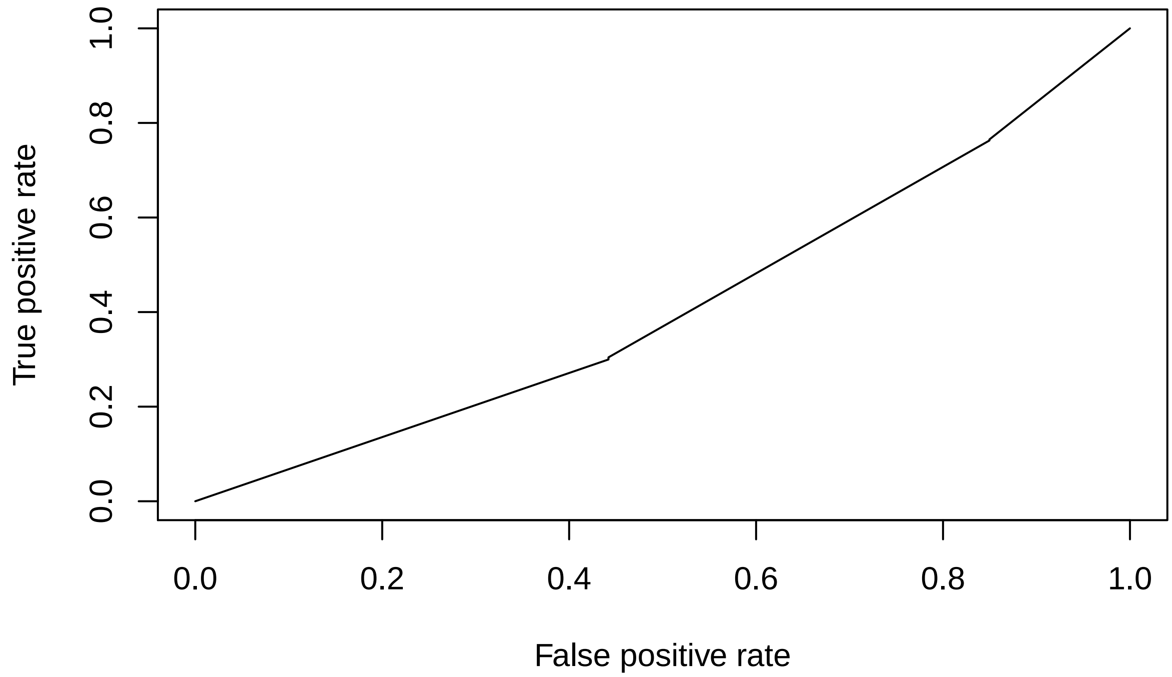
Try k=4

```
my.classifier(audit4, cl.name = "knn", do.cv = T, n = 4)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education Marital Occupation   Income Gender Deductions
## 1  38          2         3       5         12  81838.0      1          0
## 2  35          2         1       1         14  72099.0      2          0
## 3  32          2         5       2          2 154676.7      2          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test knn : #training: 1138 #testing 759
## error rate: 0.7009223
```




```
## 10 -fold CV run 1 knn : #training: 1708 #testing 189
##      error= 0.7089947
## 10 -fold CV run 2 knn : #training: 1707 #testing 190
##      error= 0.7263158
## 10 -fold CV run 3 knn : #training: 1707 #testing 190
##      error= 0.7157895
## 10 -fold CV run 4 knn : #training: 1707 #testing 190
##      error= 0.6736842
## 10 -fold CV run 5 knn : #training: 1707 #testing 190
##      error= 0.6947368
## 10 -fold CV run 6 knn : #training: 1707 #testing 190
##      error= 0.7368421
## 10 -fold CV run 7 knn : #training: 1707 #testing 190
##      error= 0.7
## 10 -fold CV run 8 knn : #training: 1707 #testing 190
##      error= 0.6894737
## 10 -fold CV run 9 knn : #training: 1708 #testing 189
##      error= 0.6931217
## 10 -fold CV run 10 knn : #training: 1708 #testing 189
##      error= 0.7089947
## 10 -fold CV results: avg error= NA
```

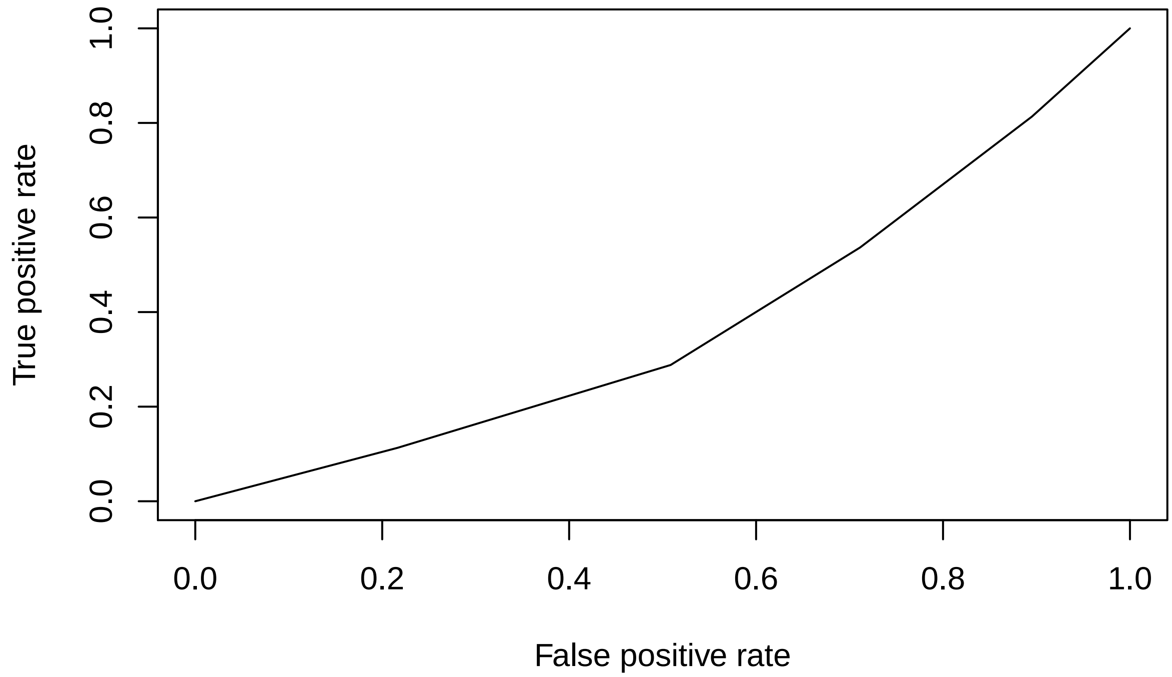



```
## error= 0.5691443 precision= 0.2035269 recall= 0.5219985 f-score 0.2998008
## auc= 0.4164183
```

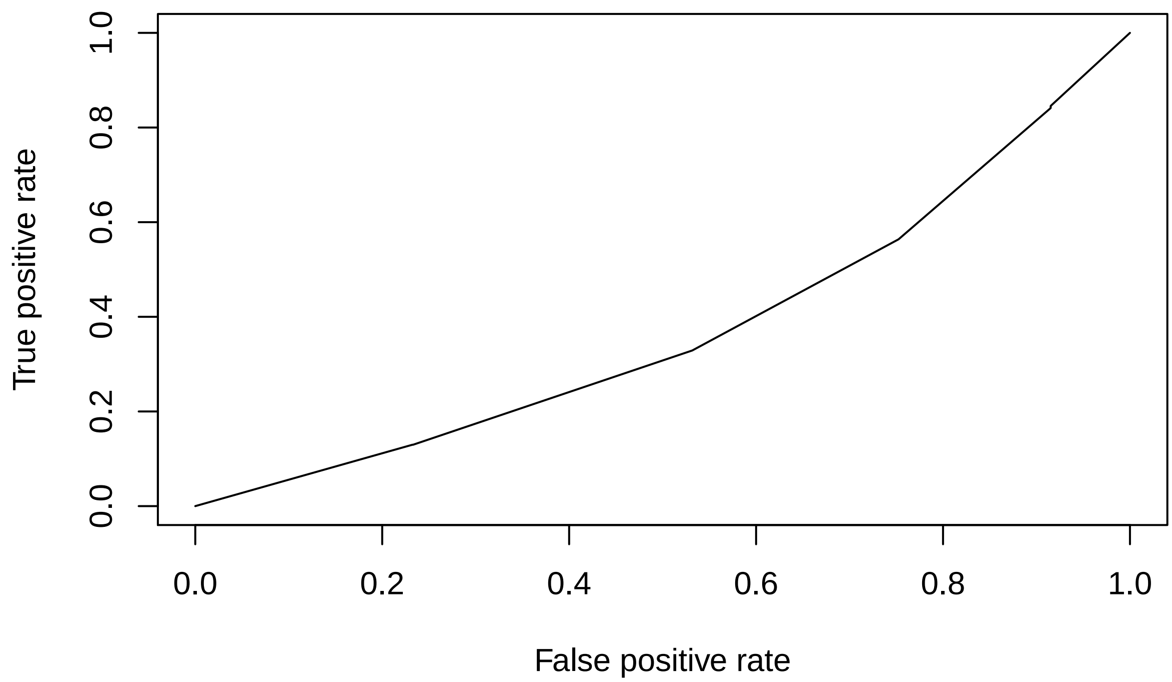
Try k=8

```r
my.classifier(audit4, cl.name = "knn", do.cv = T, n = 8)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education Marital Occupation   Income Gender Deductions
## 1  38          2         3       5         12  81838.0      1          0
## 2  35          2         1       1         14  72099.0      2          0
## 3  32          2         5       2          2 154676.7      2          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test knn : #training: 1138 #testing 759
## error rate: 0.7299078
```




```
## 10 -fold CV run 1 knn : #training: 1708 #testing 189
##      error= 0.7248677
## 10 -fold CV run 2 knn : #training: 1707 #testing 190
##      error= 0.7263158
## 10 -fold CV run 3 knn : #training: 1707 #testing 190
##      error= 0.7315789
## 10 -fold CV run 4 knn : #training: 1707 #testing 190
##      error= 0.7526316
## 10 -fold CV run 5 knn : #training: 1707 #testing 190
##      error= 0.7052632
## 10 -fold CV run 6 knn : #training: 1707 #testing 190
##      error= 0.7894737
## 10 -fold CV run 7 knn : #training: 1707 #testing 190
##      error= 0.7368421
## 10 -fold CV run 8 knn : #training: 1707 #testing 190
##      error= 0.7578947
## 10 -fold CV run 9 knn : #training: 1708 #testing 189
##      error= 0.7037037
## 10 -fold CV run 10 knn : #training: 1708 #testing 189
##      error= 0.7301587
## 10 -fold CV results: avg error= NA
```



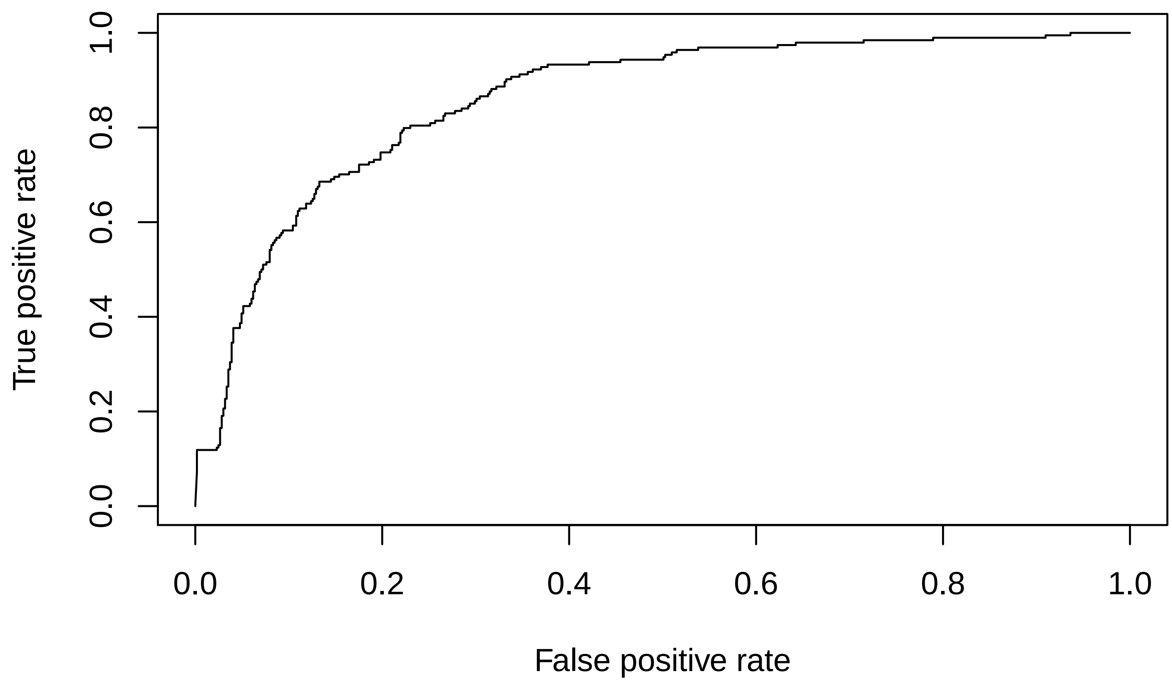

```
## error= 0.5601608 precision= 0.1884562 recall= 0.4798658 f-score 0.2649697
## auc= 0.3747512
```

When k=2, we can get the highest AUC.

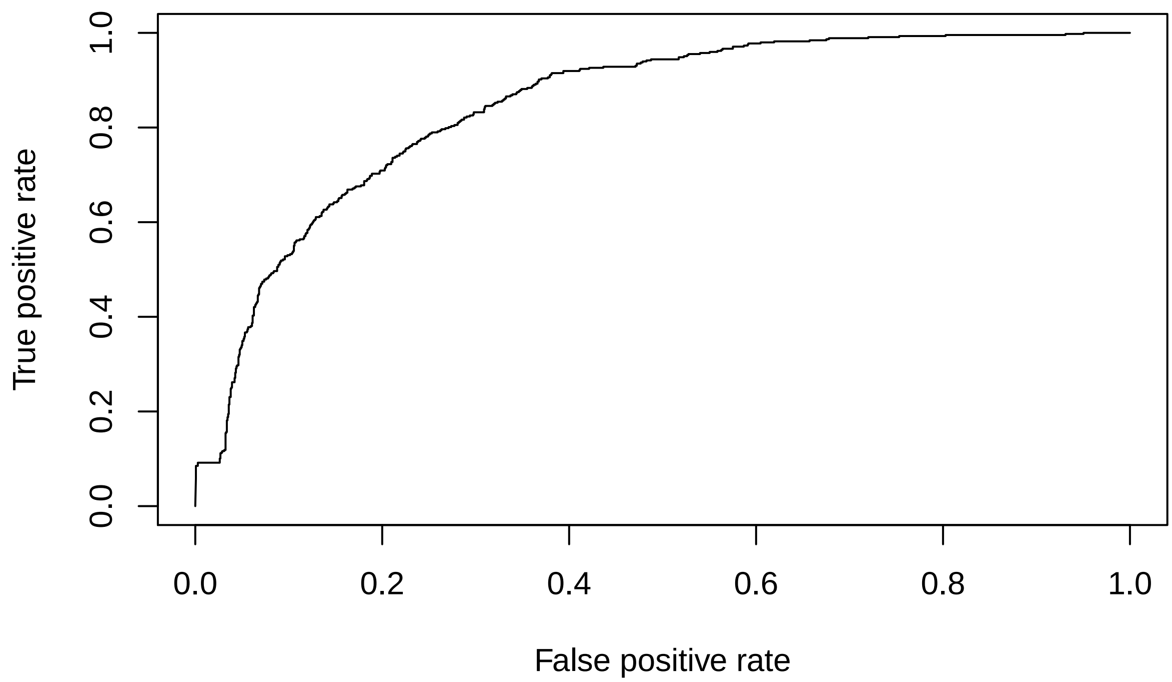
Naive Bayesian

```r
my.classifier(audit3, cl.name = "nb", do.cv = T)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education   Marital Occupation   Income Gender Deductions
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test nb : #training: 1138 #testing 759
## error rate: 0.1805007
```




```
## 10 -fold CV run 1 nb : #training: 1708 #testing 189
##      error= 0.1640212
## 10 -fold CV run 2 nb : #training: 1707 #testing 190
##      error= 0.1631579
## 10 -fold CV run 3 nb : #training: 1707 #testing 190
##      error= 0.2157895
## 10 -fold CV run 4 nb : #training: 1707 #testing 190
##      error= 0.1842105
## 10 -fold CV run 5 nb : #training: 1707 #testing 190
##      error= 0.2157895
## 10 -fold CV run 6 nb : #training: 1707 #testing 190
##      error= 0.1631579
## 10 -fold CV run 7 nb : #training: 1707 #testing 190
##      error= 0.2
## 10 -fold CV run 8 nb : #training: 1707 #testing 190
##      error= 0.1789474
## 10 -fold CV run 9 nb : #training: 1708 #testing 189
##      error= 0.1534392
## 10 -fold CV run 10 nb : #training: 1708 #testing 189
##      error= 0.2010582
## 10 -fold CV results: avg error= NA
```

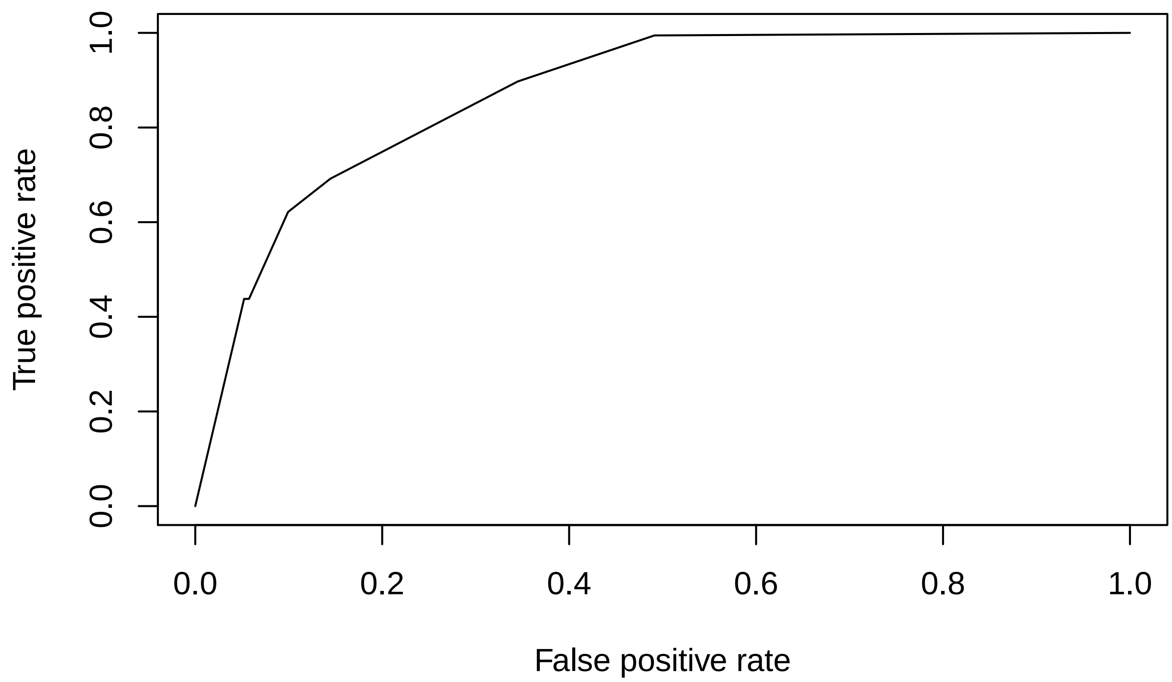



```
## error= 0.37804 precision= 0.439863 recall= 0.7761018 f-score 0.4951875
## auc= 0.8455712
```

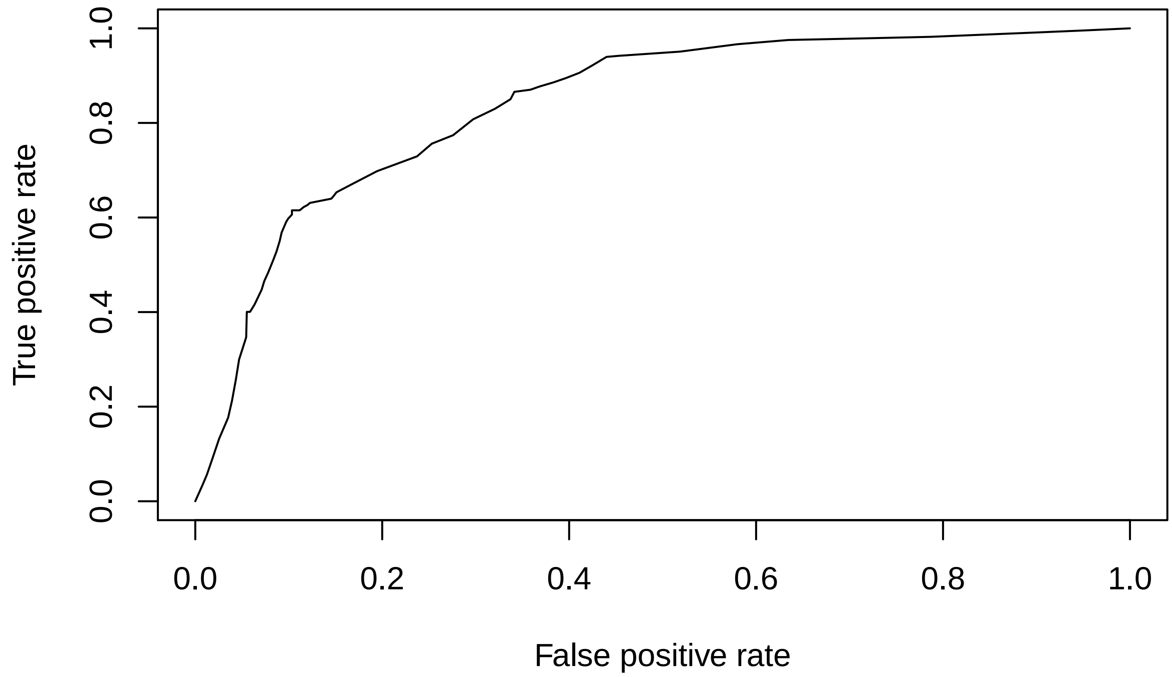
Desicion tree with default error

```
my.classifier(audit3, cl.name = "dtree", do.cv = T)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education   Marital Occupation   Income Gender Deductions
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test dtree : #training: 1138 #testing 759
## error rate: 0.1673254
```




```
## 10 -fold CV run 1 dtree : #training: 1708 #testing 189
##      error= 0.1957672
## 10 -fold CV run 2 dtree : #training: 1707 #testing 190
##      error= 0.1526316
## 10 -fold CV run 3 dtree : #training: 1707 #testing 190
##      error= 0.1631579
## 10 -fold CV run 4 dtree : #training: 1707 #testing 190
##      error= 0.2105263
## 10 -fold CV run 5 dtree : #training: 1707 #testing 190
##      error= 0.1473684
## 10 -fold CV run 6 dtree : #training: 1707 #testing 190
##      error= 0.1684211
## 10 -fold CV run 7 dtree : #training: 1707 #testing 190
##      error= 0.1894737
## 10 -fold CV run 8 dtree : #training: 1707 #testing 190
##      error= 0.1368421
## 10 -fold CV run 9 dtree : #training: 1708 #testing 189
##      error= 0.1375661
## 10 -fold CV run 10 dtree : #training: 1708 #testing 189
##      error= 0.2063492
## 10 -fold CV results: avg error= NA
```




```
## error= 0.2790107 precision= 0.5277636 recall= 0.6314318 f-score 0.5129637
## auc= 0.8396351
```

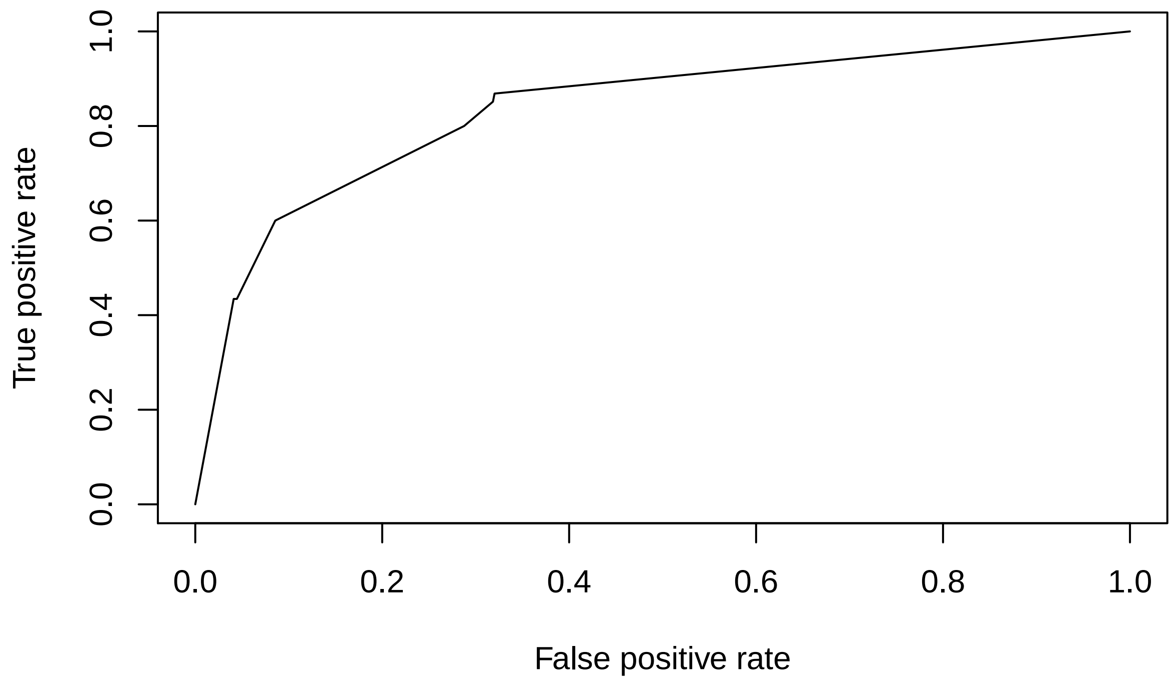
Desicion tree with prune tree (Maximum error)

```
do.classification <- function(train.set, test.set, cl.name, n, verbose = F) {
    switch(cl.name, knn = {
        prob = knn(train.set[, -10], test.set[, -10], cl = train.set[, 10],
            k = n, prob = T)
        prob = attr(prob, "prob")
        attr(prob, "prob")[prob == 0] = 1 - attr(prob, "prob")[prob == 0]
        prob
    }, lr = {
        names(train.set)

        model = glm(y ~ ., family = binomial, data = train.set)
        if (verbose) {
            print(summary(model))
        }
        prob = predict(model, newdata = test.set, type = "response")
        prob
    }, nb = {

        model = naiveBayes(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "raw")
        prob = prob[, 2]/rowSums(prob)
        prob
```

```
    }, dtree = {
        model = rpart(y ~ ., data = train.set)
        test.set = test.set[, -10]
        if (verbose) {
            print(summary(model))
            printcp(model)
            plotcp(model)
            plot(model, uniform = TRUE, main = "Classification Tree")
            text(model, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        prob = predict(model, newdata = test.set)

        if (0) {
            pfit <- prune(model, cp = model$cptable[which.max(model$cptable[,
                "xerror"]), "CP"])
            prob = predict(pfit, newdata = test.set)
            ## plot the pruned tree
            plot(pfit, uniform = TRUE, main = "Pruned Classification Tree")
            text(pfit, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        head(prob)
        prob
    }, svm = {
        model = svm(y ~ ., data = train.set, probability = T)
        if (0) {
            tuned <- tune.svm(y ~ ., data = train.set, kernel = "radial", gamma = 10^(-6:-1),
                cost = 10^(-1:2))
            summary(tuned)
            gamma = tuned[["best.parameters"]]$gamma
            cost = tuned[["best.parameters"]]$cost
            model = svm(y ~ ., data = train.set, probability = T, kernel = "radial",
                gamma = gamma, cost = cost)
        }
        test.set = test.set[, -10]
        prob = predict(model, newdata = test.set, probability = T)
        dim(prob)
        prob
    }, ada = {
        model = ada(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "probs")
        prob = prob[, 2]/rowSums(prob)
        prob
    })
}
my.classifier(audit3, cl.name = "dtree", do.cv = T)
```

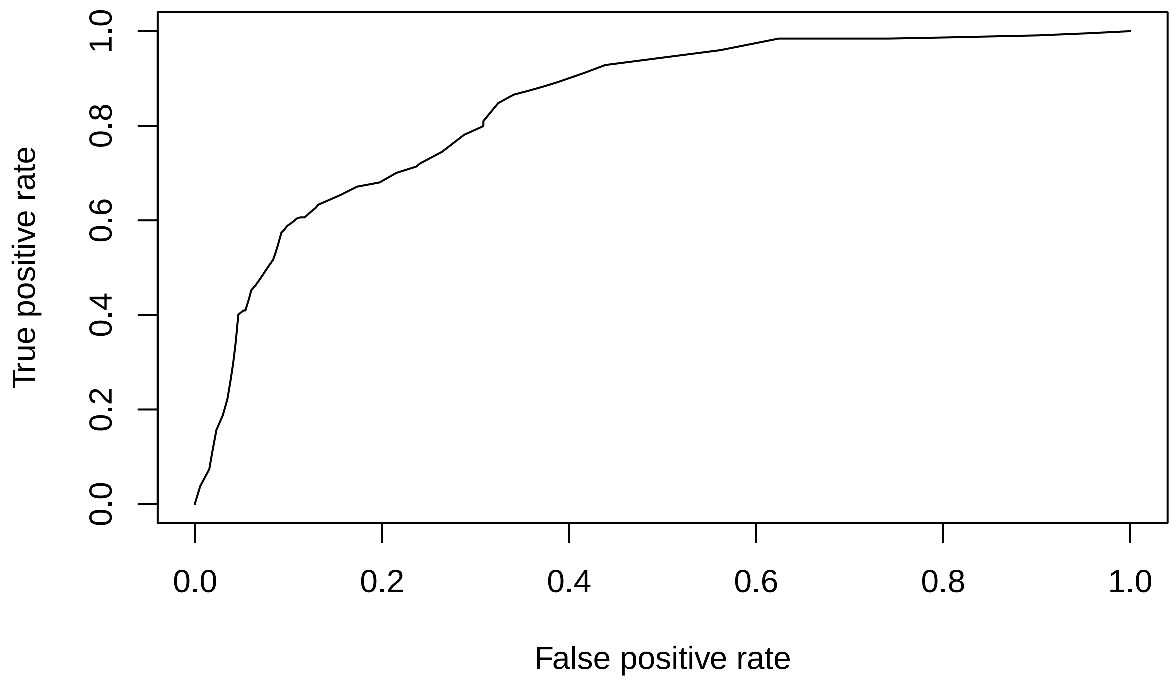
```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education   Marital Occupation   Income Gender Deductions
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours y
## 1    72 0
## 2    30 0
```

```
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test dtree : #training: 1138 #testing 759
## error rate: 0.1581028
```




```
## 10 -fold CV run 1 dtree : #training: 1708 #testing 189
##      error= 0.1164021
## 10 -fold CV run 2 dtree : #training: 1707 #testing 190
##      error= 0.1947368
## 10 -fold CV run 3 dtree : #training: 1707 #testing 190
##      error= 0.1736842
## 10 -fold CV run 4 dtree : #training: 1707 #testing 190
##      error= 0.1789474
## 10 -fold CV run 5 dtree : #training: 1707 #testing 190
##      error= 0.1736842
## 10 -fold CV run 6 dtree : #training: 1707 #testing 190
##      error= 0.1684211
## 10 -fold CV run 7 dtree : #training: 1707 #testing 190
##      error= 0.1526316
## 10 -fold CV run 8 dtree : #training: 1707 #testing 190
##      error= 0.1157895
## 10 -fold CV run 9 dtree : #training: 1708 #testing 189
##      error= 0.2063492
## 10 -fold CV run 10 dtree : #training: 1708 #testing 189
##      error= 0.2328042
```

```
## 10 -fold CV results: avg error= NA
```



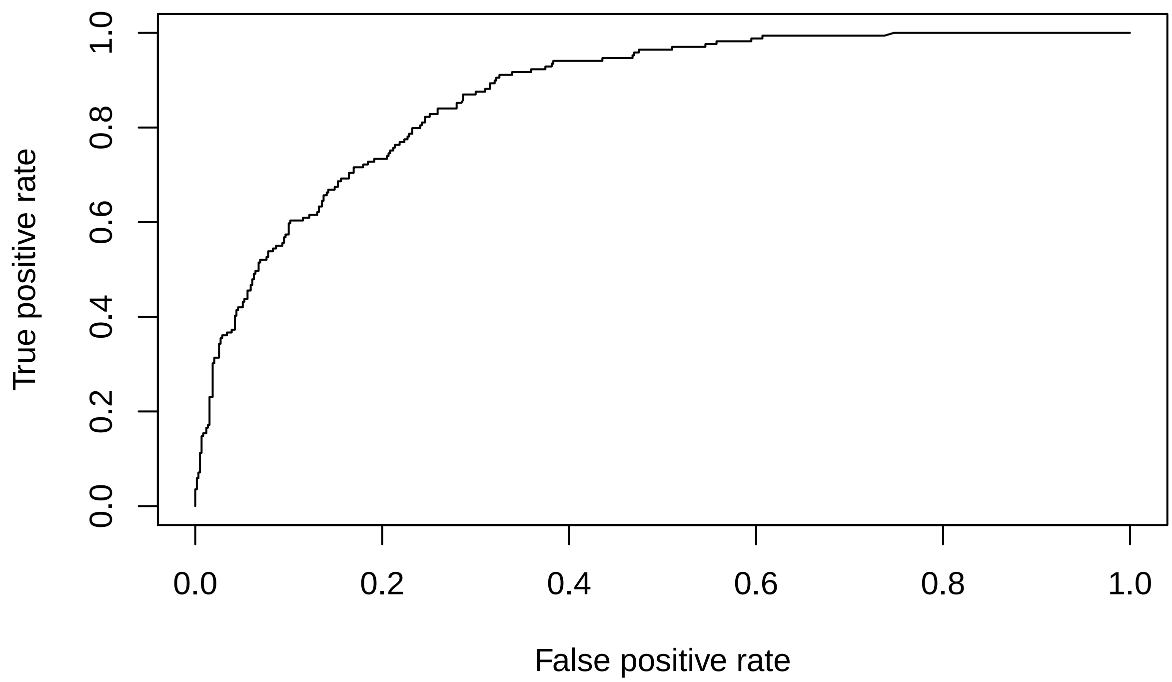

```
## error= 0.2735284 precision= 0.5492736 recall= 0.6092095 f-score 0.5056606
## auc= 0.8415027
```

Minimum error shows better AUC performance than the maximum error.

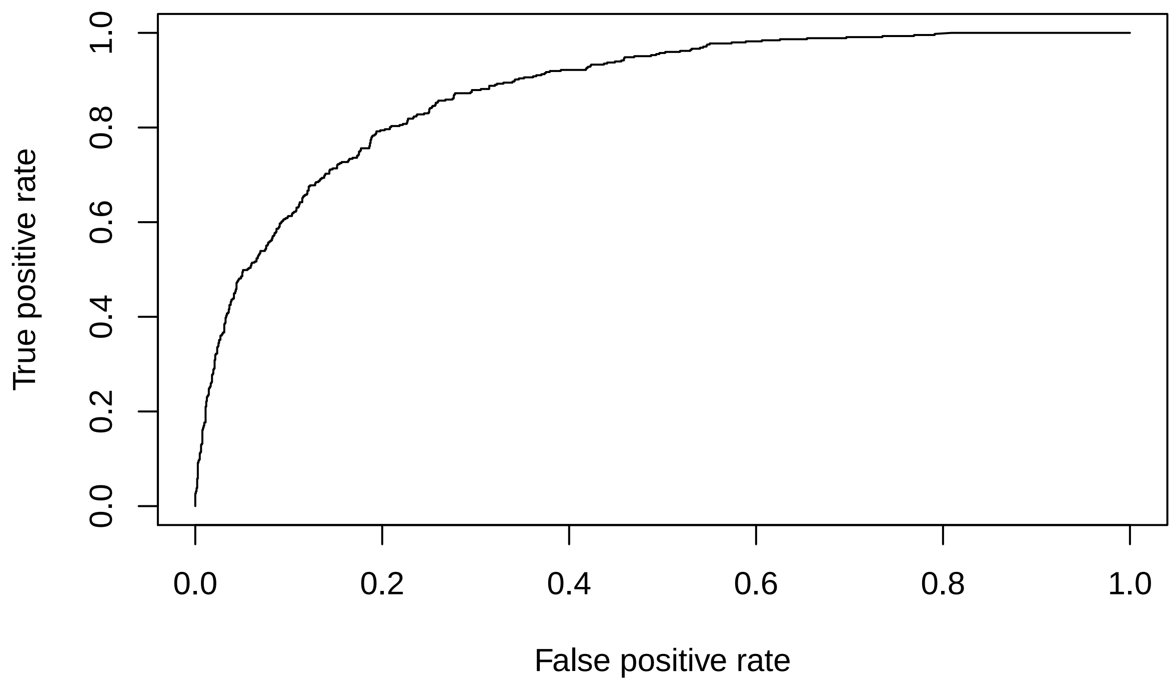
Adaboost

```
my.classifier(audit3, cl.name = "ada", do.cv = T)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education   Marital Occupation   Income Gender Deductions
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test ada : #training: 1138 #testing 759
## error rate: 0.1660079
```




```
## 10 -fold CV run 1 ada : #training: 1708 #testing 189
##      error= 0.1428571
## 10 -fold CV run 2 ada : #training: 1707 #testing 190
##      error= 0.2052632
## 10 -fold CV run 3 ada : #training: 1707 #testing 190
##      error= 0.1263158
## 10 -fold CV run 4 ada : #training: 1707 #testing 190
##      error= 0.1842105
## 10 -fold CV run 5 ada : #training: 1707 #testing 190
##      error= 0.1842105
## 10 -fold CV run 6 ada : #training: 1707 #testing 190
##      error= 0.1315789
## 10 -fold CV run 7 ada : #training: 1707 #testing 190
##      error= 0.1421053
## 10 -fold CV run 8 ada : #training: 1707 #testing 190
##      error= 0.1947368
## 10 -fold CV run 9 ada : #training: 1708 #testing 189
##      error= 0.1693122
## 10 -fold CV run 10 ada : #training: 1708 #testing 189
##      error= 0.1693122
## 10 -fold CV results: avg error= NA
```

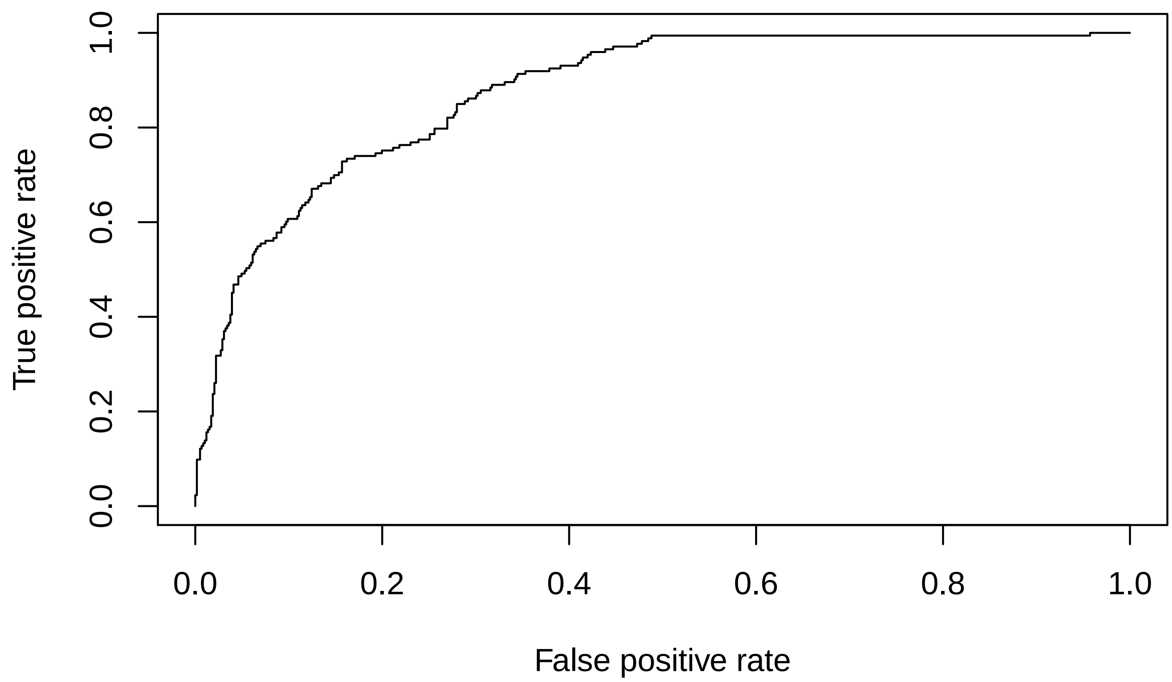



```
## error= 0.3023611 precision= 0.5265476 recall= 0.7463053 f-score 0.5327048
## auc= 0.8772028
```

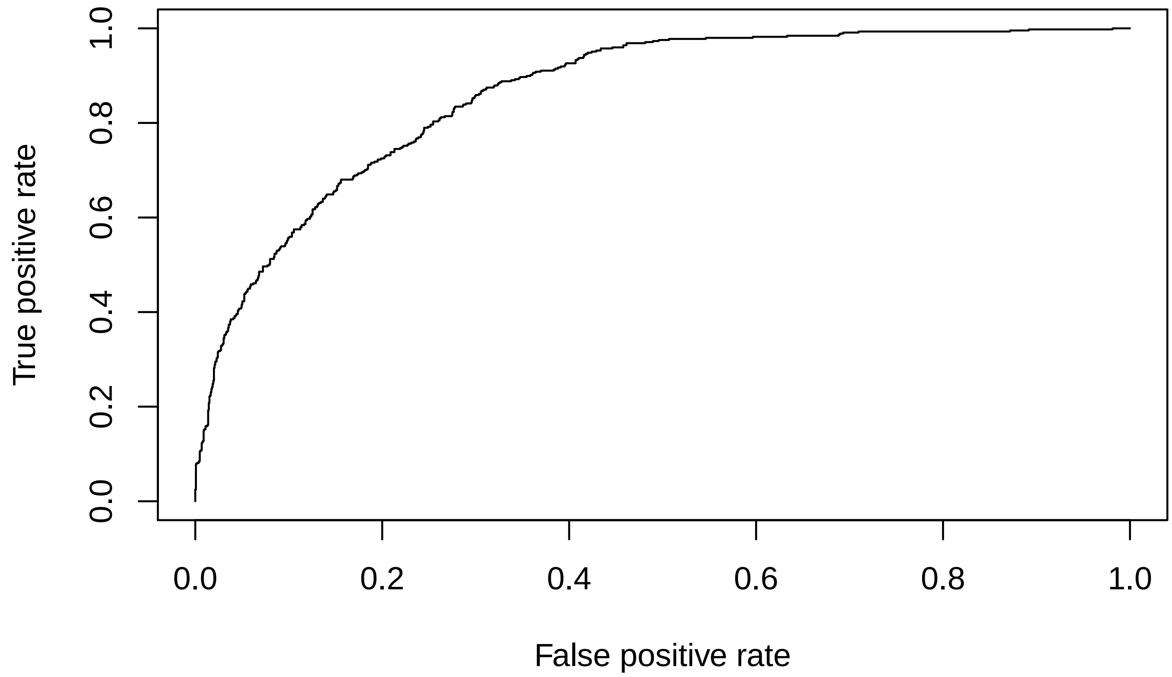
SVM with kernel="radial" gamma = 10^(-6:-1), cost = 10^(-1:2))

```
my.classifier(audit3, cl.name = "svm", do.cv = T)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education   Marital Occupation   Income Gender Deductions
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test svm : #training: 1138 #testing 759
## error rate: 0.2081686
```




```
## 10 -fold CV run 1 svm : #training: 1708 #testing 189
##      error= 0.1693122
## 10 -fold CV run 2 svm : #training: 1707 #testing 190
##      error= 0.2631579
## 10 -fold CV run 3 svm : #training: 1707 #testing 190
##      error= 0.2368421
## 10 -fold CV run 4 svm : #training: 1707 #testing 190
##      error= 0.2157895
## 10 -fold CV run 5 svm : #training: 1707 #testing 190
##      error= 0.2315789
## 10 -fold CV run 6 svm : #training: 1707 #testing 190
##      error= 0.1736842
## 10 -fold CV run 7 svm : #training: 1707 #testing 190
##      error= 0.2368421
## 10 -fold CV run 8 svm : #training: 1707 #testing 190
##      error= 0.2105263
## 10 -fold CV run 9 svm : #training: 1708 #testing 189
##      error= 0.1957672
## 10 -fold CV run 10 svm : #training: 1708 #testing 189
##      error= 0.2275132
## 10 -fold CV results: avg error= NA
```




```
## error= 0.36946 precision= 0.4704614 recall= 0.7769959 f-score 0.501625
## auc= 0.8625781
```

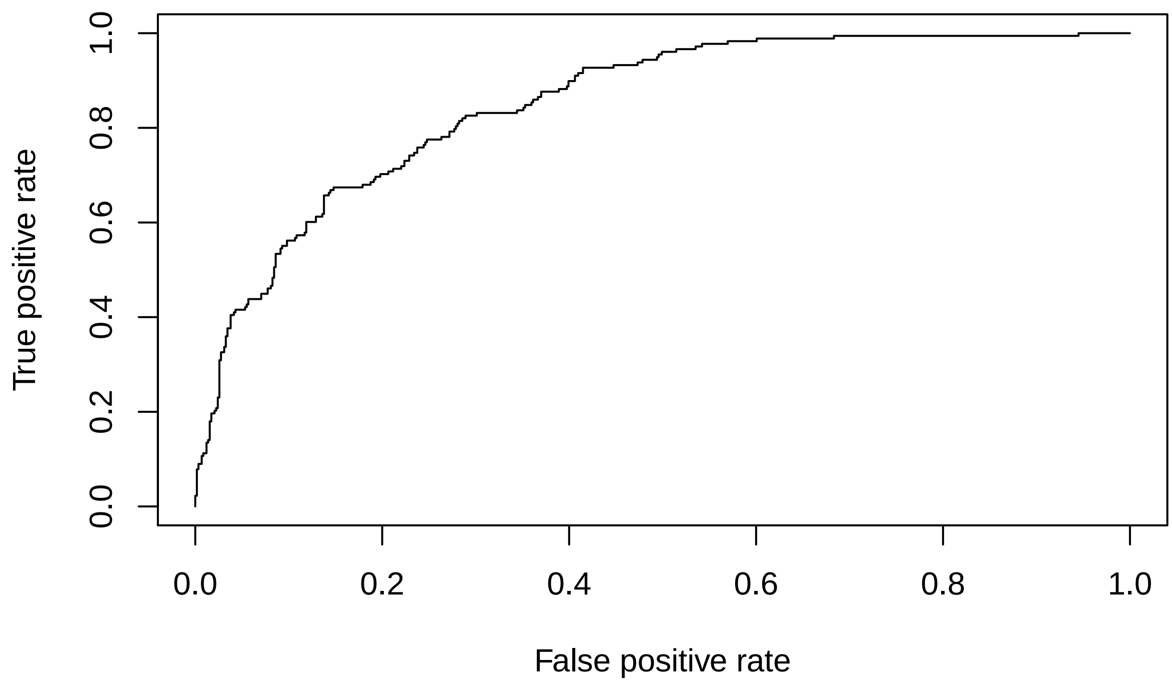
SVM with kernel="linear" gamma = 10^(-6:-1), cost = 10^(1:2))

```
do.classification <- function(train.set, test.set, cl.name, n, verbose = F) {
    switch(cl.name, knn = {
        prob = knn(train.set[, -10], test.set[, -10], cl = train.set[, 10],
            k = n, prob = T)
        prob = attr(prob, "prob")
        attr(prob, "prob")[prob == 0] = 1 - attr(prob, "prob")[prob == 0]
        prob
    }, lr = {
        # logistic regression
        names(train.set)

        model = glm(y ~ ., family = binomial, data = train.set)
        if (verbose) {
            print(summary(model))
        }
        prob = predict(model, newdata = test.set, type = "response")
        prob
    }, nb = {

        model = naiveBayes(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "raw")
        prob = prob[, 2]/rowSums(prob)
```

```
        prob
    }, dtree = {
        model = rpart(y ~ ., data = train.set)
        test.set = test.set[, -10]
        if (verbose) {
            print(summary(model))
            printcp(model)
            plotcp(model)
            ## plot the tree
            plot(model, uniform = TRUE, main = "Classification Tree")
            text(model, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        prob = predict(model, newdata = test.set)

        if (0) {
            pfit <- prune(model, cp = model$cptable[which.min(model$cptable[,
                "xerror"]), "CP"])
            prob = predict(pfit, newdata = test.set)
            ## plot the pruned tree
            plot(pfit, uniform = TRUE, main = "Pruned Classification Tree")
            text(pfit, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        head(prob)
        prob
    }, svm = {
        model = svm(y ~ ., data = train.set, probability = T)
        if (0) {
            tuned <- tune.svm(y ~ ., data = train.set, kernel = "linear", gamma = 10^(-6:-1),
                cost = 10^(1:2))
            summary(tuned)
            gamma = tuned[["best.parameters"]]$gamma
            cost = tuned[["best.parameters"]]$cost
            model = svm(y ~ ., data = train.set, probability = T, kernel = "radial",
                gamma = gamma, cost = cost)
        }
        test.set = test.set[, -10]
        prob = predict(model, newdata = test.set, probability = T)
        dim(prob)
        prob
    }, ada = {
        model = ada(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "probs")
        prob = prob[, 2]/rowSums(prob)
        prob
    })
}
my.classifier(audit3, cl.name = "svm", do.cv = T)
```

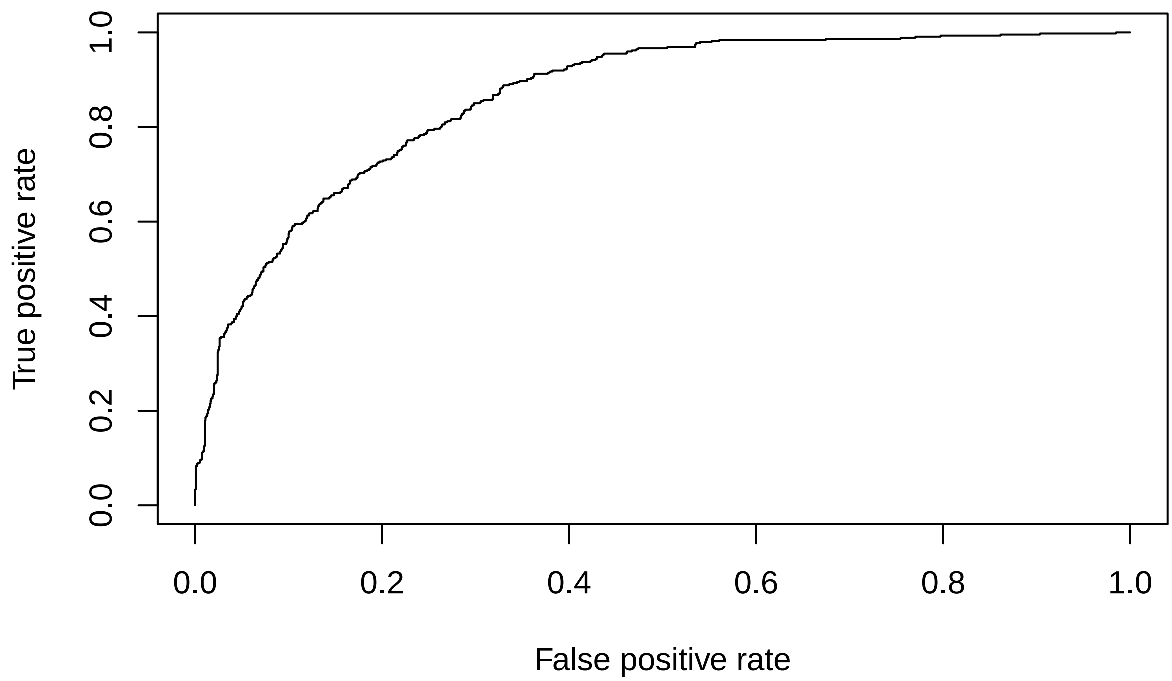
```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education   Marital Occupation   Income Gender Deductions
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours y
```

```
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test svm : #training: 1138 #testing 759
## error rate: 0.2173913
```




```
## 10 -fold CV run 1 svm : #training: 1708 #testing 189
##      error= 0.1693122
## 10 -fold CV run 2 svm : #training: 1707 #testing 190
##      error= 0.2631579
## 10 -fold CV run 3 svm : #training: 1707 #testing 190
##      error= 0.2157895
## 10 -fold CV run 4 svm : #training: 1707 #testing 190
##      error= 0.2052632
## 10 -fold CV run 5 svm : #training: 1707 #testing 190
##      error= 0.2263158
## 10 -fold CV run 6 svm : #training: 1707 #testing 190
##      error= 0.2368421
## 10 -fold CV run 7 svm : #training: 1707 #testing 190
##      error= 0.2
## 10 -fold CV run 8 svm : #training: 1707 #testing 190
##      error= 0.2263158
## 10 -fold CV run 9 svm : #training: 1708 #testing 189
##      error= 0.2486772
```

```
## 10 -fold CV run 10 svm : #training: 1708 #testing 189
##      error= 0.1904762
## 10 -fold CV results: avg error= NA
```




```
## error= 0.3696394 precision= 0.4707906 recall= 0.7766152 f-score 0.5017037
## auc= 0.8620798
```

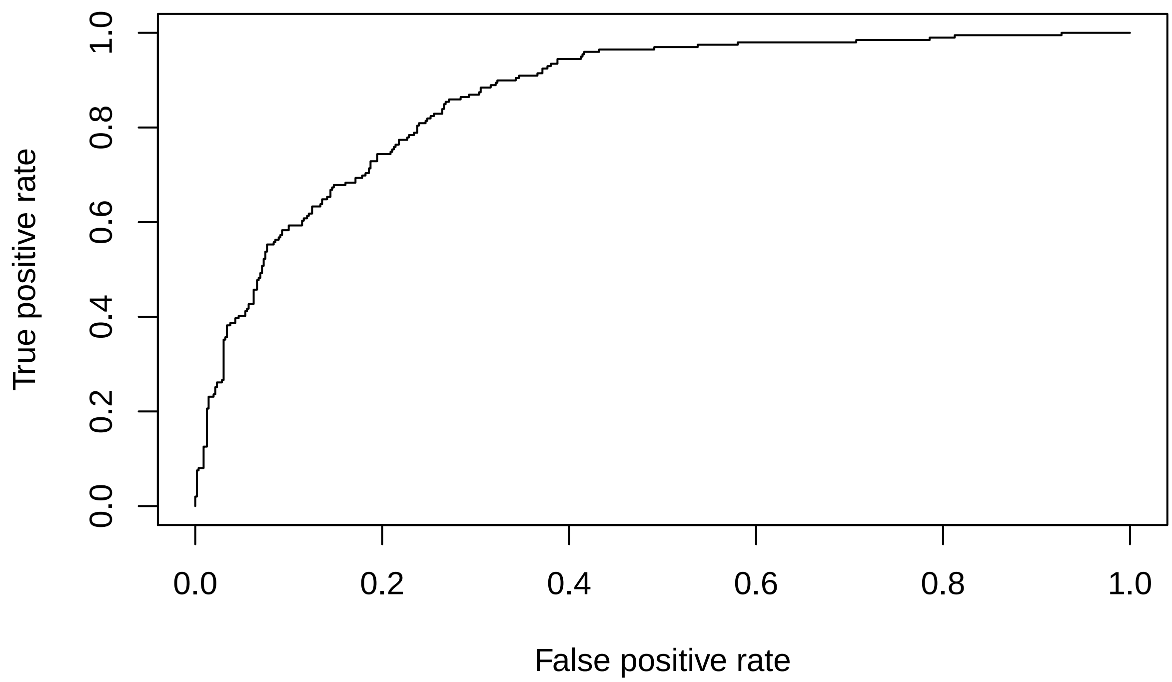
SVM with kernel="linear" gamma = 10^(-4:-1), cost = 10^(1:2))

```
do.classification <- function(train.set, test.set, cl.name, n, verbose = F) {
    switch(cl.name, knn = {
        prob = knn(train.set[, -10], test.set[, -10], cl = train.set[, 10],
            k = n, prob = T)
        prob = attr(prob, "prob")
        attr(prob, "prob")[prob == 0] = 1 - attr(prob, "prob")[prob == 0]
        prob
    }, lr = {
        names(train.set)

        model = glm(y ~ ., family = binomial, data = train.set)
        if (verbose) {
            print(summary(model))
        }
        prob = predict(model, newdata = test.set, type = "response")
        prob
    }, nb = {
```

```
        model = naiveBayes(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "raw")
        prob = prob[, 2]/rowSums(prob)
        prob
    }, dtree = {
        model = rpart(y ~ ., data = train.set)
        test.set = test.set[, -10]
        if (verbose) {
            print(summary(model))
            printcp(model)
            plotcp(model)
            ## plot the tree
            plot(model, uniform = TRUE, main = "Classification Tree")
            text(model, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        prob = predict(model, newdata = test.set)

        if (0) {
            pfit <- prune(model, cp = model$cptable[which.min(model$cptable[,
                "xerror"]), "CP"])
            prob = predict(pfit, newdata = test.set)
            ## plot the pruned tree
            plot(pfit, uniform = TRUE, main = "Pruned Classification Tree")
            text(pfit, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        head(prob)
        prob
    }, svm = {
        model = svm(y ~ ., data = train.set, probability = T)
        if (0) {
            tuned <- tune.svm(y ~ ., data = train.set, kernel = "linear", gamma = 10^(-4:-1),
                cost = 10^(1:2))
            summary(tuned)
            gamma = tuned[["best.parameters"]]$gamma
            cost = tuned[["best.parameters"]]$cost
            model = svm(y ~ ., data = train.set, probability = T, kernel = "radial",
                gamma = gamma, cost = cost)
        }
        test.set = test.set[, -10]
        prob = predict(model, newdata = test.set, probability = T)
        dim(prob)
        prob
    }, ada = {
        model = ada(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "probs")
        prob = prob[, 2]/rowSums(prob)
        prob
    })
}
my.classifier(audit3, cl.name = "svm", do.cv = T)
```

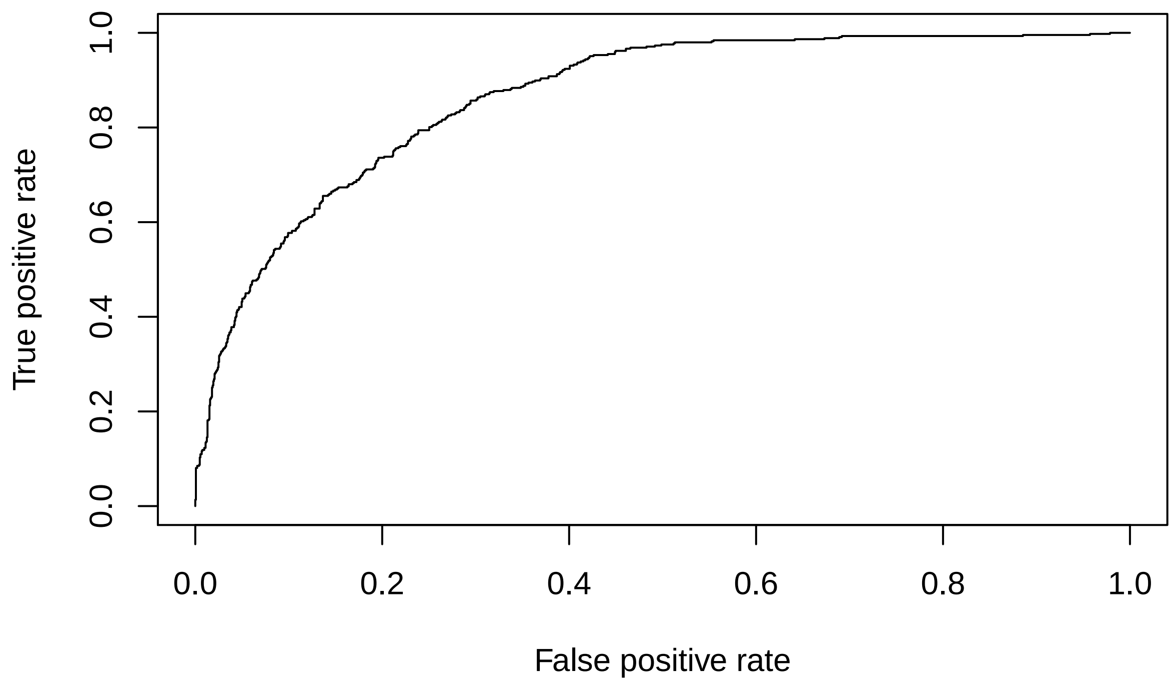
```
## my dataset: 1897 observations 9 predictors
##    Age Employment Education   Marital Occupation   Income Gender Deductions
## 1   38    Private   College Unmarried    Service  81838.0 Female          0
```

```
## 2  35    Private Associate    Absent  Transport  72099.0   Male           0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male           0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test svm : #training: 1138 #testing 759
## error rate: 0.2437418
```




```
## 10 -fold CV run 1 svm : #training: 1708 #testing 189
##      error= 0.2063492
## 10 -fold CV run 2 svm : #training: 1707 #testing 190
##      error= 0.2
## 10 -fold CV run 3 svm : #training: 1707 #testing 190
##      error= 0.1894737
## 10 -fold CV run 4 svm : #training: 1707 #testing 190
##      error= 0.1842105
## 10 -fold CV run 5 svm : #training: 1707 #testing 190
##      error= 0.2315789
## 10 -fold CV run 6 svm : #training: 1707 #testing 190
##      error= 0.2210526
## 10 -fold CV run 7 svm : #training: 1707 #testing 190
##      error= 0.1684211
## 10 -fold CV run 8 svm : #training: 1707 #testing 190
```

```
##      error= 0.2105263
## 10 -fold CV run 9 svm : #training: 1708 #testing 189
##      error= 0.2857143
## 10 -fold CV run 10 svm : #training: 1708 #testing 189
##      error= 0.2698413
## 10 -fold CV results: avg error= NA
```



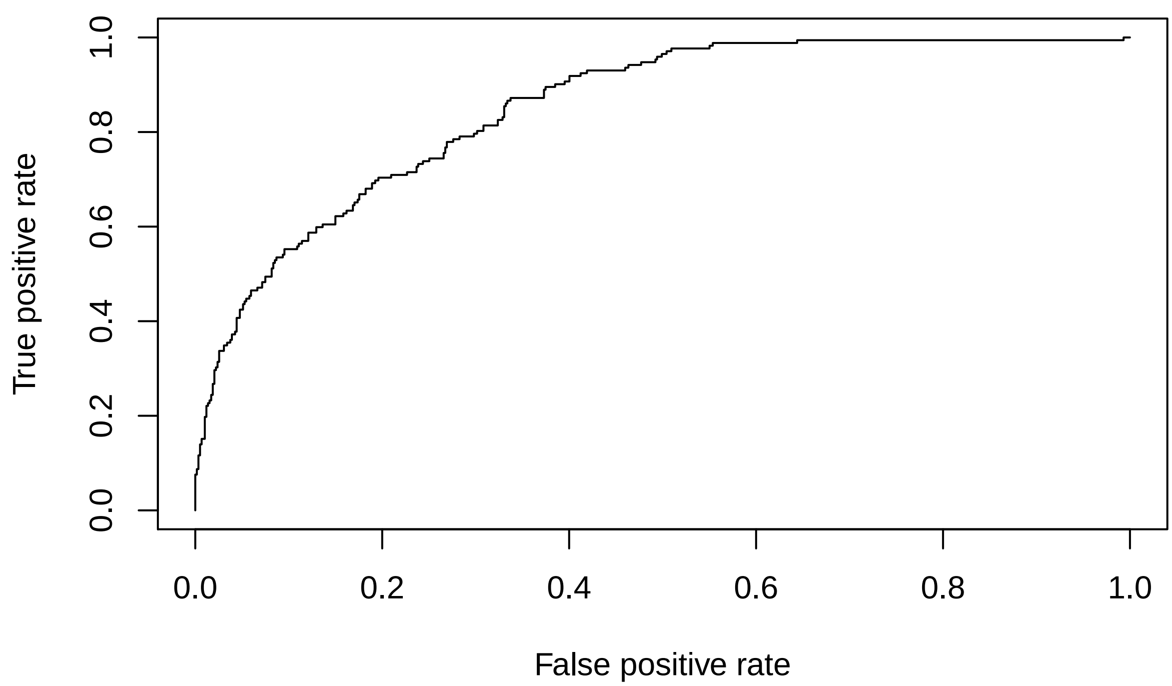

```
## error= 0.3688295 precision= 0.4709408 recall= 0.7783337 f-score 0.5027936
## auc= 0.8643292
```

SVM with kernel="polynomial" gamma = 10^(-4:-1), cost = 10^(1:2))

```
do.classification <- function(train.set, test.set, cl.name, n, verbose = F) {
    switch(cl.name, knn = {
        prob = knn(train.set[, -10], test.set[, -10], cl = train.set[, 10],
            k = n, prob = T)
        prob = attr(prob, "prob")
        attr(prob, "prob")[prob == 0] = 1 - attr(prob, "prob")[prob == 0]
        prob
    }, lr = {
        names(train.set)

        model = glm(y ~ ., family = binomial, data = train.set)
        if (verbose) {
            print(summary(model))
        }
        prob = predict(model, newdata = test.set, type = "response")
```

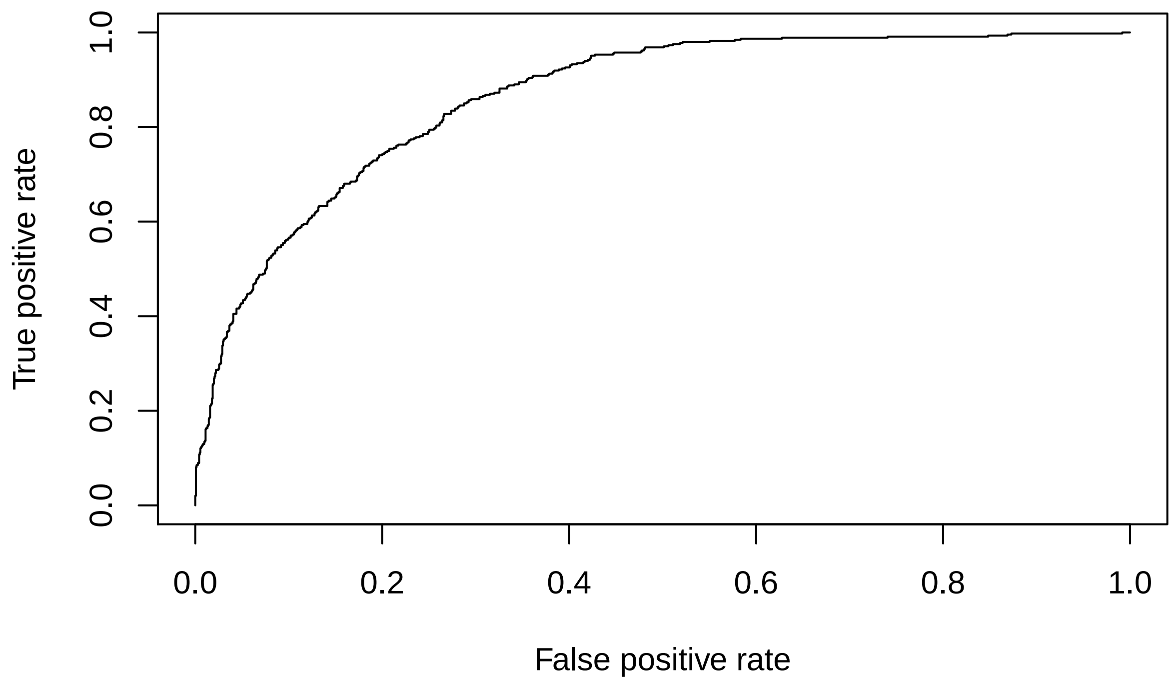
```r
        prob
    }, nb = {

        model = naiveBayes(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "raw")
        prob = prob[, 2]/rowSums(prob)
        prob
    }, dtree = {
        model = rpart(y ~ ., data = train.set)
        test.set = test.set[, -10]
        if (verbose) {
            print(summary(model))
            printcp(model)
            plotcp(model)
            ## plot the tree
            plot(model, uniform = TRUE, main = "Classification Tree")
            text(model, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        prob = predict(model, newdata = test.set)

        if (0) {
            pfit <- prune(model, cp = model$cptable[which.min(model$cptable[,
                "xerror"]), "CP"])
            prob = predict(pfit, newdata = test.set)
            ## plot the pruned tree
            plot(pfit, uniform = TRUE, main = "Pruned Classification Tree")
            text(pfit, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        head(prob)
        prob
    }, svm = {
        model = svm(y ~ ., data = train.set, probability = T)
        if (0) {
            tuned <- tune.svm(y ~ ., data = train.set, kernel = "polynomial",
                gamma = 10^(-4:-1), cost = 10^(1:2))
            summary(tuned)
            gamma = tuned[["best.parameters"]]$gamma
            cost = tuned[["best.parameters"]]$cost
            model = svm(y ~ ., data = train.set, probability = T, kernel = "radial",
                gamma = gamma, cost = cost)
        }
        test.set = test.set[, -10]
        prob = predict(model, newdata = test.set, probability = T)
        dim(prob)
        prob
    }, ada = {
        model = ada(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "probs")
        prob = prob[, 2]/rowSums(prob)
        prob
    })
}
my.classifier(audit3, cl.name = "svm", do.cv = T)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education   Marital Occupation   Income Gender Deductions
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test svm : #training: 1138 #testing 759
## error rate: 0.2094862
```




```
## 10 -fold CV run 1 svm : #training: 1708 #testing 189
##      error= 0.2275132
## 10 -fold CV run 2 svm : #training: 1707 #testing 190
##      error= 0.2105263
## 10 -fold CV run 3 svm : #training: 1707 #testing 190
##      error= 0.2315789
## 10 -fold CV run 4 svm : #training: 1707 #testing 190
##      error= 0.2473684
## 10 -fold CV run 5 svm : #training: 1707 #testing 190
##      error= 0.1947368
## 10 -fold CV run 6 svm : #training: 1707 #testing 190
##      error= 0.1842105
```

```
## 10 -fold CV run 7 svm : #training: 1707 #testing 190
##      error= 0.1736842
## 10 -fold CV run 8 svm : #training: 1707 #testing 190
##      error= 0.2578947
## 10 -fold CV run 9 svm : #training: 1708 #testing 189
##      error= 0.2328042
## 10 -fold CV run 10 svm : #training: 1708 #testing 189
##      error= 0.1904762
## 10 -fold CV results: avg error= NA
```




```
## error= 0.3689784 precision= 0.4712798 recall= 0.7780178 f-score 0.5025565
## auc= 0.8639158
```

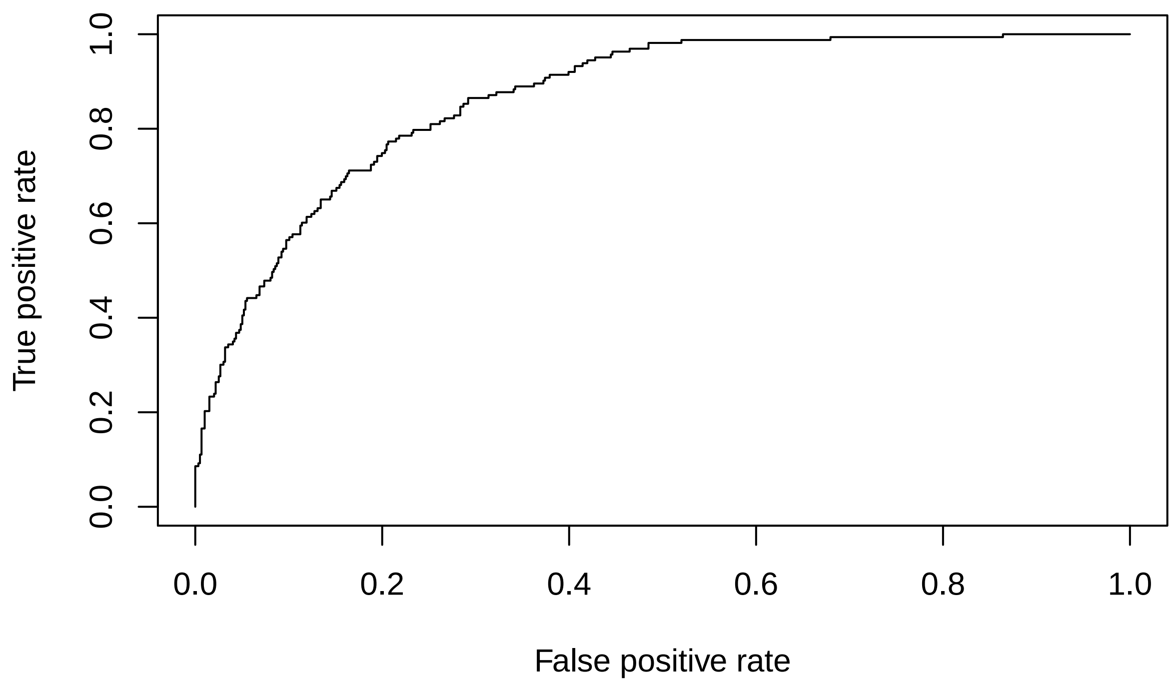
SVM with kernel="sigmoid" gamma = 10^(-4:-1), cost = 10^(1:2))

```
do.classification <- function(train.set, test.set, cl.name, n, verbose = F) {
    switch(cl.name, knn = {
        prob = knn(train.set[, -10], test.set[, -10], cl = train.set[, 10],
            k = n, prob = T)
        prob = attr(prob, "prob")
        attr(prob, "prob")[prob == 0] = 1 - attr(prob, "prob")[prob == 0]
        prob
    }, lr = {
        names(train.set)

        model = glm(y ~ ., family = binomial, data = train.set)
        if (verbose) {
```

```r
            print(summary(model))
        }
        prob = predict(model, newdata = test.set, type = "response")
        prob
    }, nb = {

        model = naiveBayes(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "raw")
        prob = prob[, 2]/rowSums(prob)
        prob
    }, dtree = {
        model = rpart(y ~ ., data = train.set)
        test.set = test.set[, -10]
        if (verbose) {
            print(summary(model))
            printcp(model)
            plotcp(model)
            ## plot the tree
            plot(model, uniform = TRUE, main = "Classification Tree")
            text(model, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        prob = predict(model, newdata = test.set)

        if (0) {
            pfit <- prune(model, cp = model$cptable[which.min(model$cptable[,
                "xerror"]), "CP"])
            prob = predict(pfit, newdata = test.set)
            ## plot the pruned tree
            plot(pfit, uniform = TRUE, main = "Pruned Classification Tree")
            text(pfit, use.n = TRUE, all = TRUE, cex = 0.8)
        }
        head(prob)
        prob
    }, svm = {
        model = svm(y ~ ., data = train.set, probability = T)
        if (0) {
            tuned <- tune.svm(y ~ ., data = train.set, kernel = "sigmoid", gamma = 10^(-4:-1),
                cost = 10^(1:2))
            summary(tuned)
            gamma = tuned[["best.parameters"]]$gamma
            cost = tuned[["best.parameters"]]$cost
            model = svm(y ~ ., data = train.set, probability = T, kernel = "radial",
                gamma = gamma, cost = cost)
        }
        test.set = test.set[, -10]
        prob = predict(model, newdata = test.set, probability = T)
        dim(prob)
        prob
    }, ada = {
        model = ada(y ~ ., data = train.set)
        prob = predict(model, newdata = test.set, type = "probs")
        prob = prob[, 2]/rowSums(prob)
        prob
```

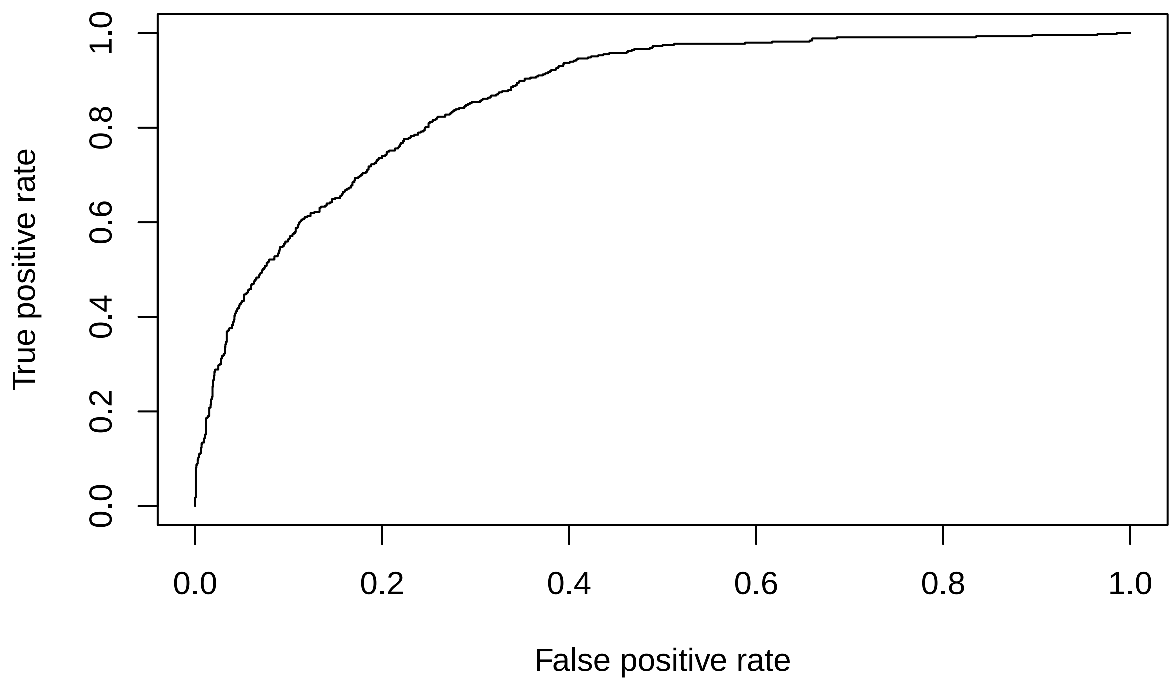
```r
    })
}
my.classifier(audit3, cl.name = "svm", do.cv = T)
```

```
## my dataset: 1897 observations 9 predictors
##   Age Employment Education   Marital Occupation   Income Gender Deductions
## 1  38    Private   College Unmarried    Service  81838.0 Female          0
## 2  35    Private Associate    Absent  Transport  72099.0   Male          0
## 3  32    Private    HSgrad  Divorced   Clerical 154676.7   Male          0
##   Hours y
## 1    72 0
## 2    30 0
## 3    40 0
## label (y) distribution:
##    0    1
## 1450  447
## pre-test svm : #training: 1138 #testing 759
## error rate: 0.1870883
```




```
## 10 -fold CV run 1 svm : #training: 1708 #testing 189
##      error= 0.2010582
## 10 -fold CV run 2 svm : #training: 1707 #testing 190
##      error= 0.2157895
## 10 -fold CV run 3 svm : #training: 1707 #testing 190
##      error= 0.1578947
## 10 -fold CV run 4 svm : #training: 1707 #testing 190
```

```
##      error= 0.2368421
## 10 -fold CV run 5 svm : #training: 1707 #testing 190
##      error= 0.2631579
## 10 -fold CV run 6 svm : #training: 1707 #testing 190
##      error= 0.2052632
## 10 -fold CV run 7 svm : #training: 1707 #testing 190
##      error= 0.2105263
## 10 -fold CV run 8 svm : #training: 1707 #testing 190
##      error= 0.1789474
## 10 -fold CV run 9 svm : #training: 1708 #testing 189
##      error= 0.2328042
## 10 -fold CV run 10 svm : #training: 1708 #testing 189
##      error= 0.2380952
## 10 -fold CV results: avg error= NA
```




```
## error= 0.3689379 precision= 0.4718327 recall= 0.7781039 f-score 0.5028076
## auc= 0.8640284
```

It shows SVM with kernel="sigmoid" gamma = 10^(-4:-1), cost = 10^(1:2)) will give us the best result.

a.Generate the table to report the values of different performance measures for each classification technique.

```
Accuracy = c(1 - 0.3642535, 1 - 0.5373397, 1 - 0.5155509, 1 - 0.5539009, 1 -
    0.3773794, 1 - 0.283924, 1 - 0.2876411, 1 - 0.3019827, 1 - 0.369064, 1 -
    0.3686135, 1 - 0.3706021, 1 - 0.3691601)
```

```r
Precision = c(0.4782371, 0.2224593, 0.213851, 0.2052566, 0.4408058, 0.5365357,
    0.5298075, 0.5276435, 0.4729202, 0.4728131, 0.469037, 0.4721092)
Recall = c(0.7880437, 0.5249814, 0.4705444, 0.5050336, 0.776951, 0.632392, 0.6562524,
    0.7504386, 0.7778363, 0.7787922, 0.7745726, 0.7776324)
F_score = c(0.5114609, 0.3441325, 0.3216605, 0.3075742, 0.4958848, 0.5095349,
    0.5210202, 0.5354407, 0.5029481, 0.5035869, 0.4998756, 0.5025245)
AUC = c(0.8770393, 0.4526444, 0.43892, 0.4043586, 0.8471411, 0.8461313, 0.8444064,
    0.8811294, 0.8636782, 0.8649294, 0.859406, 0.8634112)
result = rbind(Accuracy, Precision, Recall, F_score, AUC)
result = as.data.frame(result)
colnames(result) = c("logistic_regression", "knn-2", "knn-3", "knn-4", "NB",
    "DT-default", "DT-prune", "Adaboost", "SVM-radical", "SVM-linear", "SVM-polynomial",
    "SVM-sigmoid")

Accuracy1 = c(1 - 0.3642535, 1 - 0.5373397, 1 - 0.5155509, 1 - 0.5539009, 1 -
    0.3773794, 1 - 0.283924, 1 - 0.2876411)
Precision1 = c(0.4782371, 0.2224593, 0.213851, 0.2052566, 0.4408058, 0.5365357,
    0.5298075)
Recall1 = c(0.7880437, 0.5249814, 0.4705444, 0.5050336, 0.776951, 0.632392,
    0.6562524)
F_score1 = c(0.5114609, 0.3469871, 0.3223208, 0.2998008, 0.4958848, 0.5095349,
    0.5210202)
AUC1 = c(0.8770393, 0.4526444, 0.43892, 0.4043586, 0.8471411, 0.8461313, 0.8444064)
result1 = rbind(Accuracy1, Precision1, Recall1, F_score1, AUC1)
result1 = as.data.frame(result1)
colnames(result1) = c("logistic_regression", "knn-2", "knn-3", "knn-4", "NB",
    "DT-default", "DT-prune")

Accuracy2 = c(1 - 0.3019827, 1 - 0.369064, 1 - 0.3686135, 1 - 0.3706021, 1 -
    0.3691601)
Precision2 = c(0.5276435, 0.4729202, 0.4728131, 0.469037, 0.4721092)
Recall2 = c(0.7504386, 0.7778363, 0.7787922, 0.7745726, 0.7776324)
F_score2 = c(0.5354407, 0.3441325, 0.3216605, 0.3075742, 0.5025245)
AUC2 = c(0.8811294, 0.8636782, 0.8649294, 0.859406, 0.8634112)
result2 = rbind(Accuracy2, Precision2, Recall2, F_score2, AUC2)
result2 = as.data.frame(result2)
colnames(result2) = c("Adaboost", "SVM-radical", "SVM-linear", "SVM-polynomial",
    "SVM-sigmoid")

library(knitr)
kable(result1, caption = "Table 1: Summary of Classification")
```

Table 1: Table 1: Summary of Classification

	logistic_regression	knn-2	knn-3	knn-4	NB	DT-default	DT-prune
Accuracy1	0.6357465	0.4626603	0.4844491	0.4460991	0.6226206	0.7160760	0.7123589
Precision1	0.4782371	0.2224593	0.2138510	0.2052566	0.4408058	0.5365357	0.5298075
Recall1	0.7880437	0.5249814	0.4705444	0.5050336	0.7769510	0.6323920	0.6562524
F_score1	0.5114609	0.3469871	0.3223208	0.2998008	0.4958848	0.5095349	0.5210202
AUC1	0.8770393	0.4526444	0.4389200	0.4043586	0.8471411	0.8461313	0.8444064

```r
kable(result2, caption = "Table 2: Summary of Classification")
```

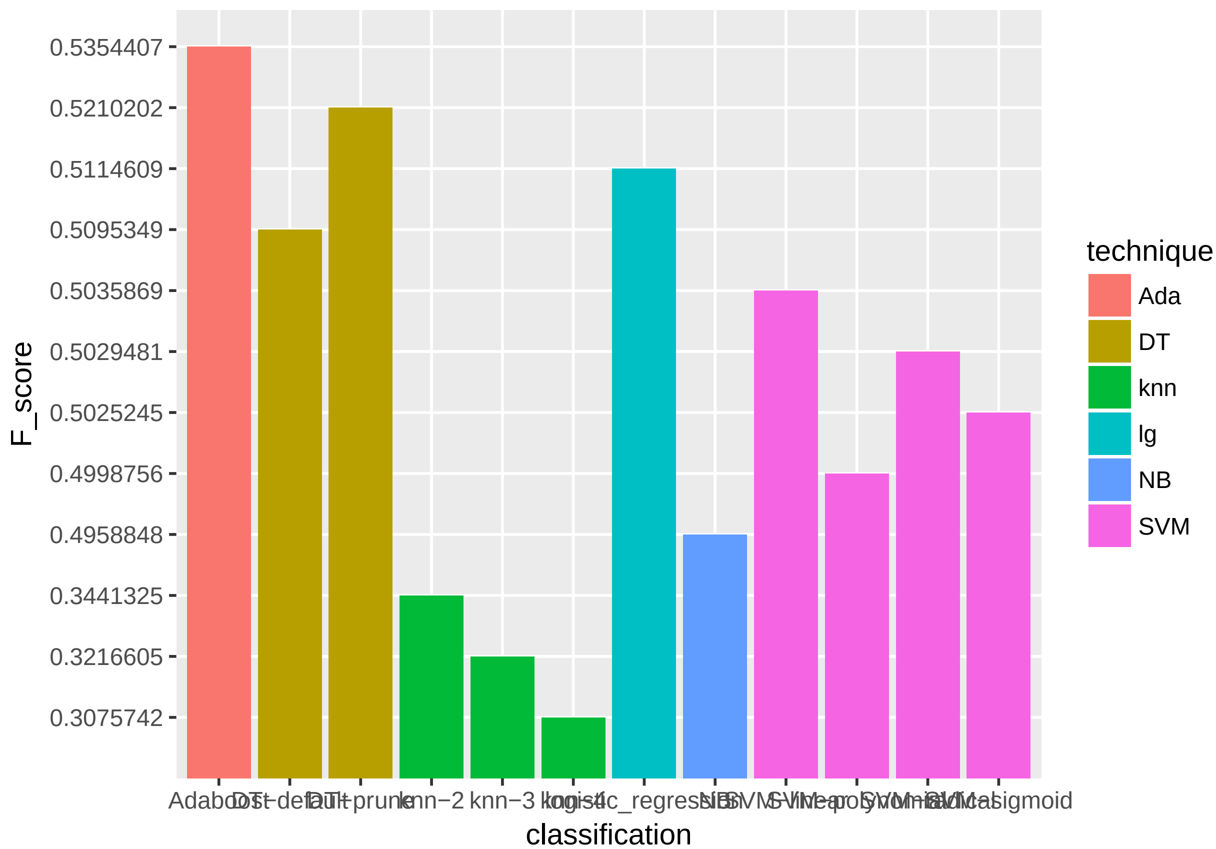
Table 2: Table 2: Summary of Classification

	Adaboost	SVM-radical	SVM-linear	SVM-polynomial	SVM-sigmoid
Accuracy2	0.6980173	0.6309360	0.6313865	0.6293979	0.6308399
Precision2	0.5276435	0.4729202	0.4728131	0.4690370	0.4721092
Recall2	0.7504386	0.7778363	0.7787922	0.7745726	0.7776324
F__score2	0.5354407	0.3441325	0.3216605	0.3075742	0.5025245
AUC2	0.8811294	0.8636782	0.8649294	0.8594060	0.8634112

b.Generate two bar charts, one for F-score and one for AUC, that allow for visually comparing different classification techniques.

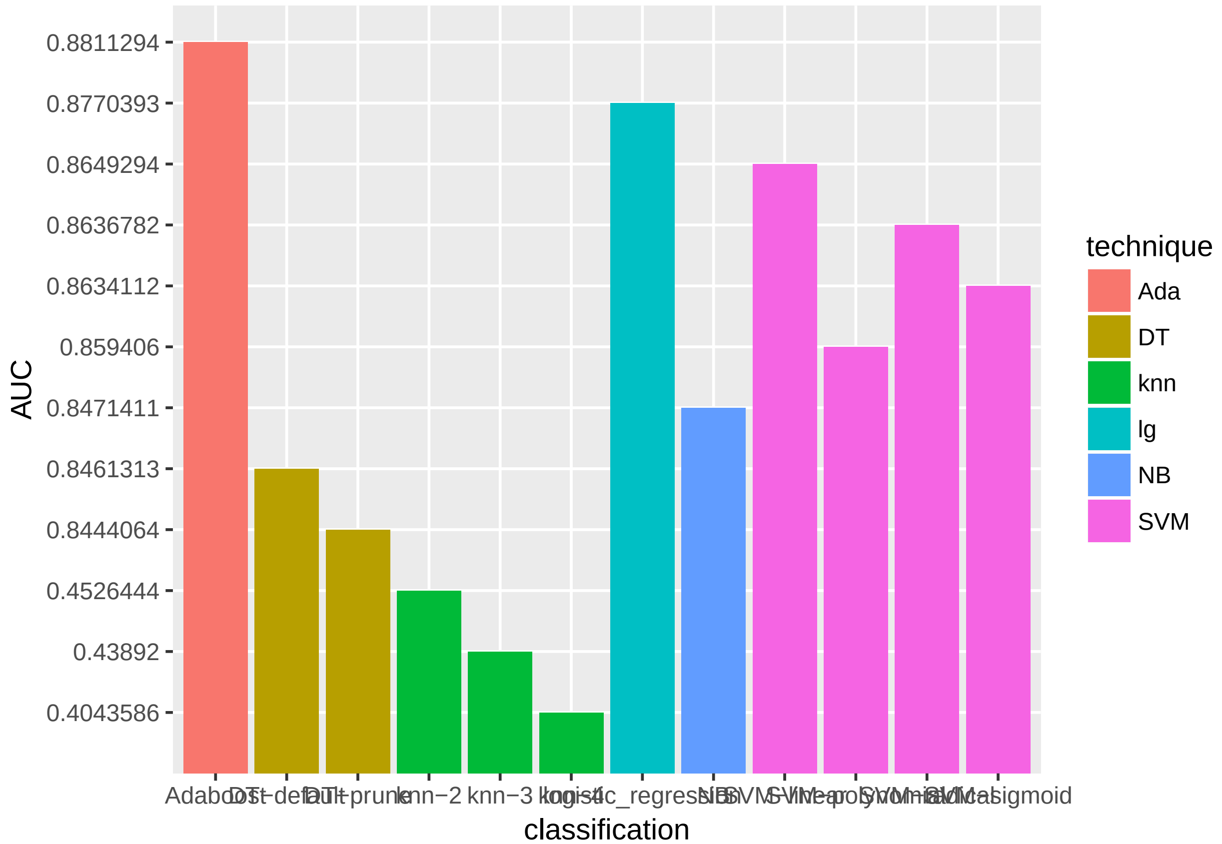
F-score bar chart

```r
library(ggplot2)
library(plyr)
library(reshape2)

classification = c("logistic_regression", "knn-2", "knn-3", "knn-4", "NB", "DT-default",
    "DT-prune", "Adaboost", "SVM-radical", "SVM-linear", "SVM-polynomial", "SVM-sigmoid")
technique = c("lg", "knn", "knn", "knn", "NB", "DT", "DT", "Ada", "SVM", "SVM",
    "SVM", "SVM")
result3 = cbind(F_score, AUC, classification, technique)
result3 = as.data.frame(result3)

ggplot(result3, aes(x = classification, y = F_score, fill = technique)) + geom_bar(stat = "identity")
```



AUC plot

```
ggplot(result3, aes(x = classification, y = AUC, fill = technique)) + geom_bar(stat = "identity")
```

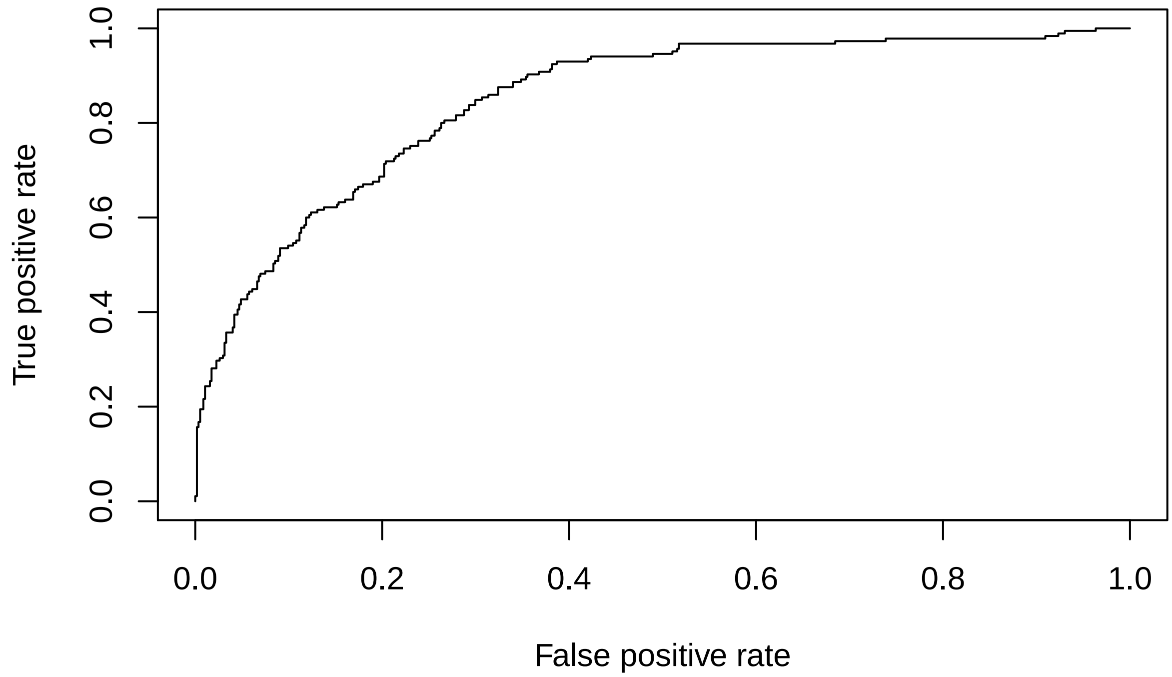


3.Generate an ROC plot that plot the ROC curve of each model into the same figure

Requirement: Generate an ROC plot that plot the ROC curve of each model into the same figure and include a legend to indicate the name of each curve. For techniques with variants, plot the best curve that has the highest AUC.

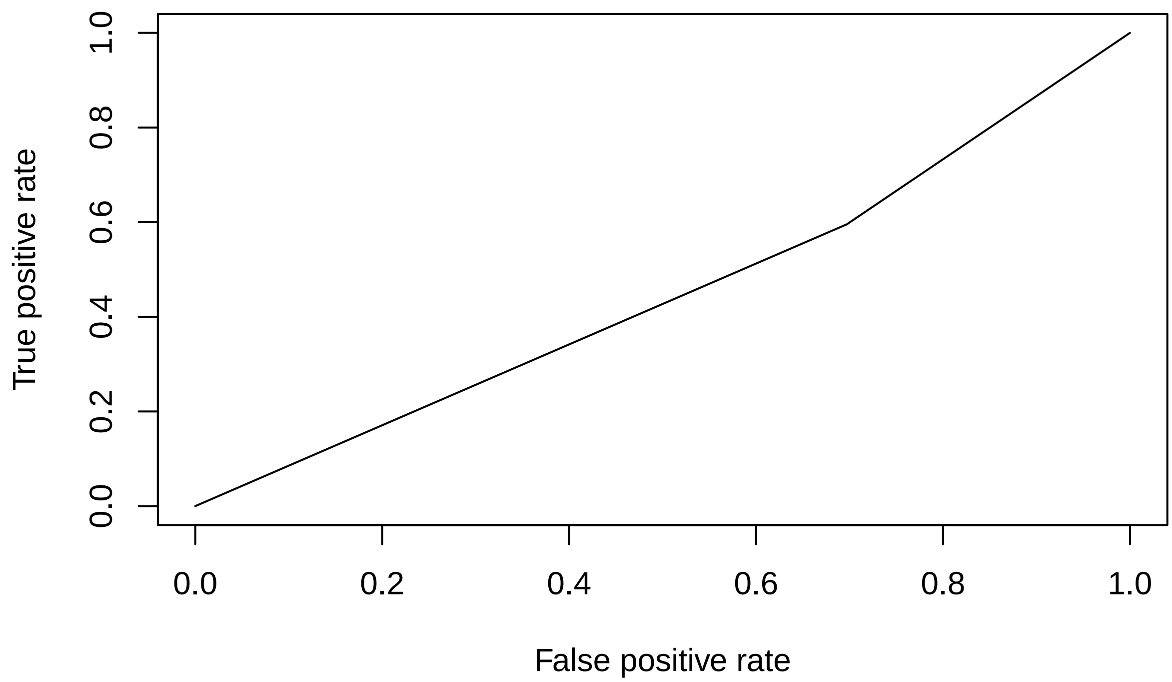
```
lrr_pred = pre.test(audit3, cl.name = "lr")
```

```
## pre-test lr : #training: 1138 #testing 759
## error rate: 0.1805007
```



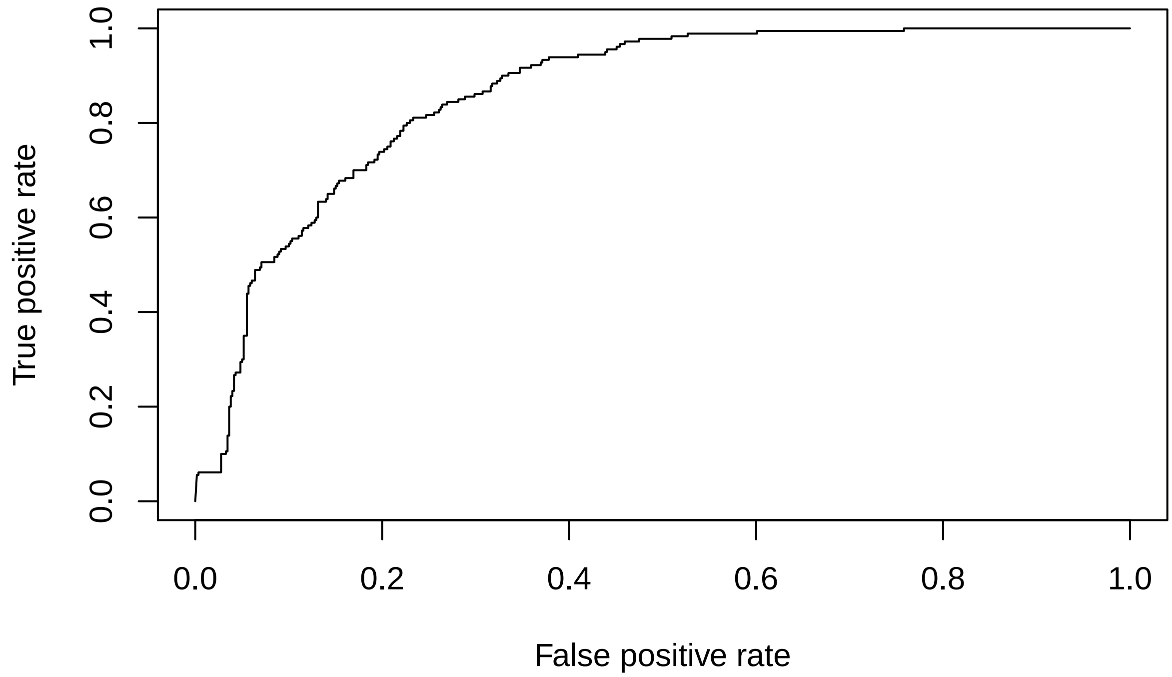

```
lrr_perf = performance(lrr_pred, "tpr", "fpr")

knn_pred = pre.test(audit4, cl.name = "knn", n = 2)
```

```
## pre-test knn : #training: 1138 #testing 759
## error rate: 0.6284585
```



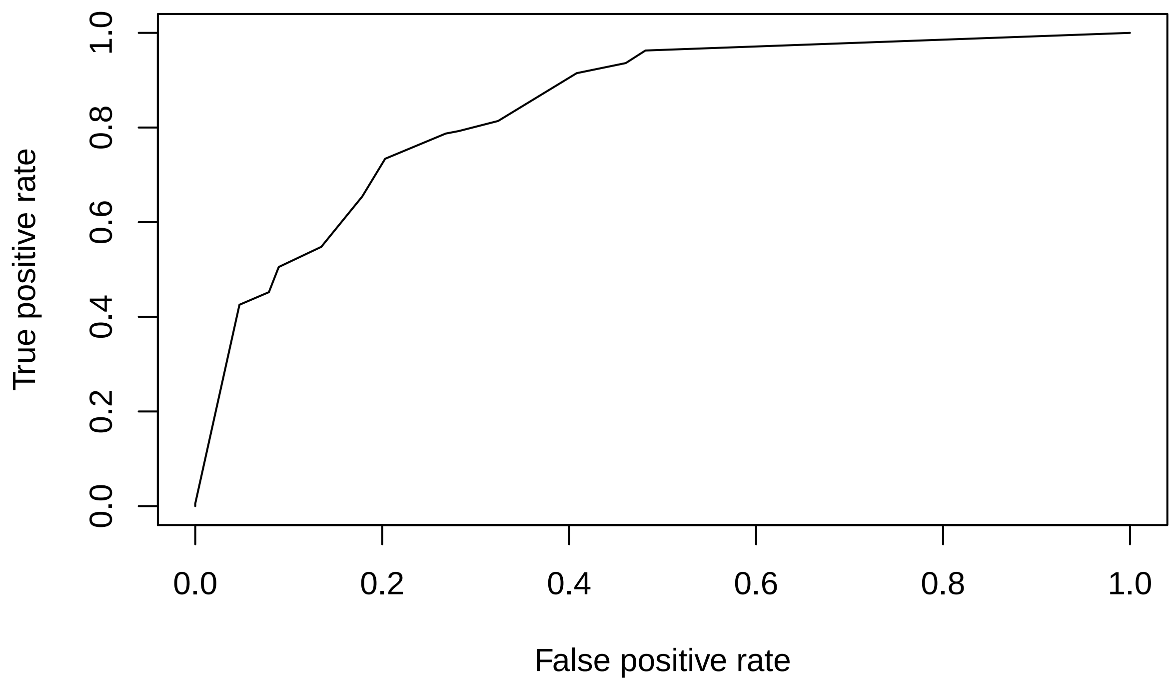

```
knn_perf = performance(knn_pred, "tpr", "fpr")

nb_pred = pre.test(audit3, cl.name = "nb")
```

```
## pre-test nb : #training: 1138 #testing 759
## error rate: 0.171278
```



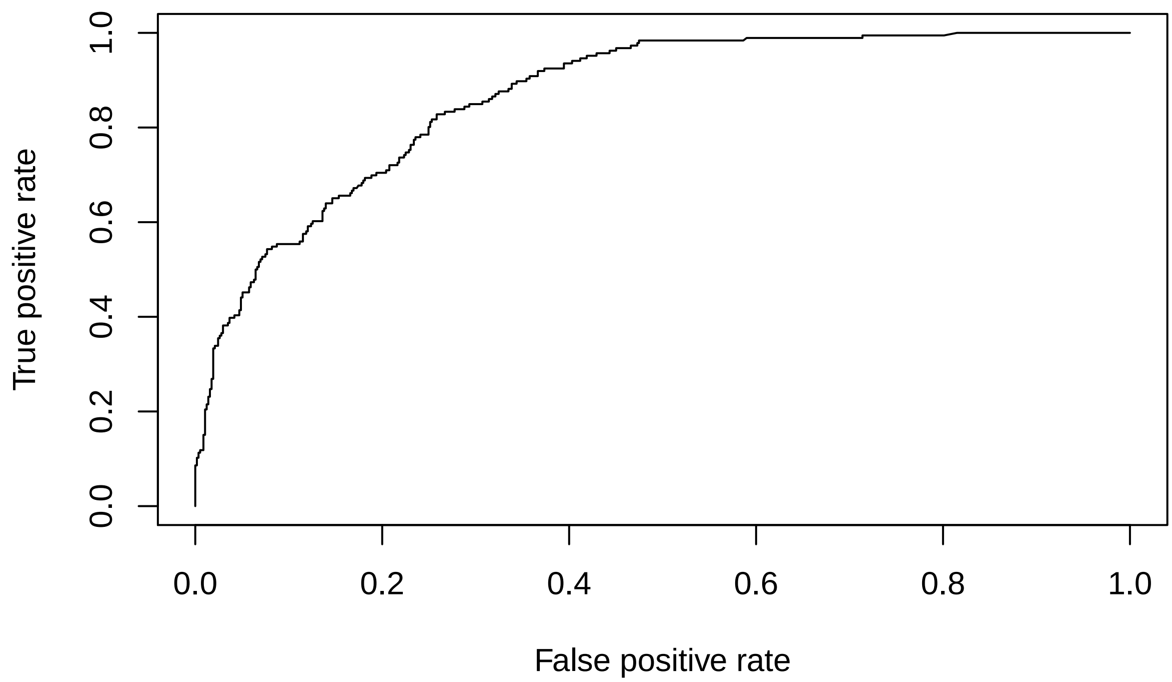

```r
nb_perf = performance(nb_pred, "tpr", "fpr")

dt_pred = pre.test(audit3, cl.name = "dtree")
```

```
## pre-test dtree : #training: 1138 #testing 759
## error rate: 0.2134387
```



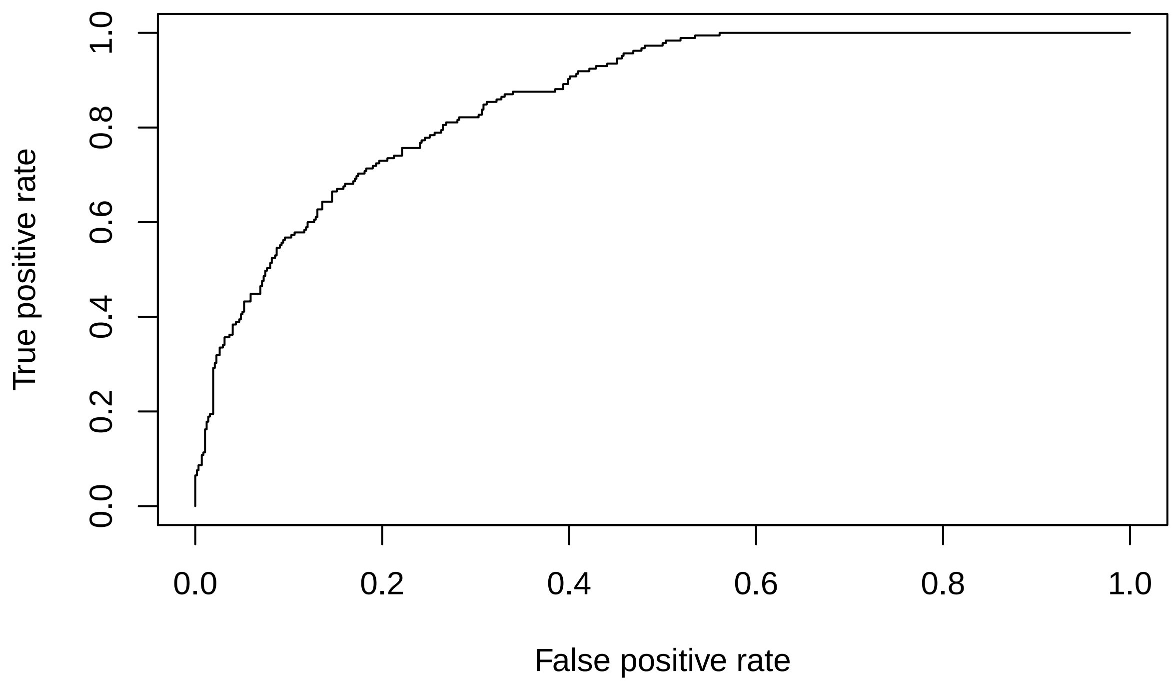

```
dt_perf = performance(dt_pred, "tpr", "fpr")

ada_pred = pre.test(audit3, cl.name = "ada")
```

```
## pre-test ada : #training: 1138 #testing 759
## error rate: 0.1699605
```

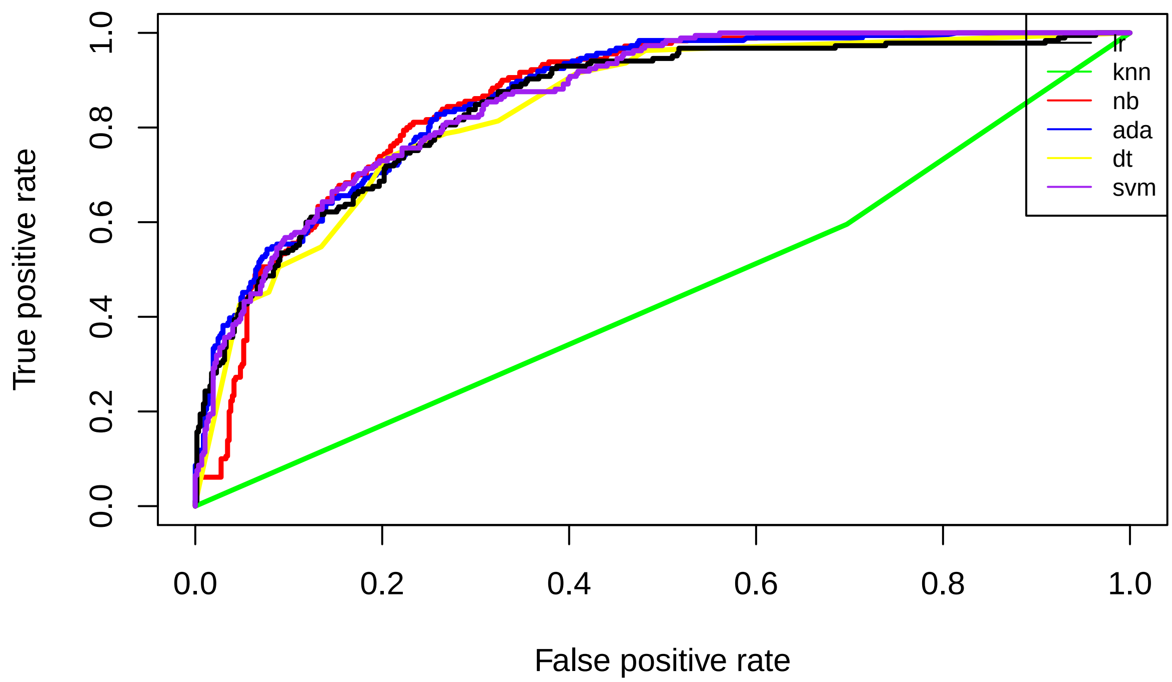



```r
ada_perf = performance(ada_pred, "tpr", "fpr")

svm_pred = pre.test(audit3, cl.name = "svm")
```

```
## pre-test svm : #training: 1138 #testing 759
## error rate: 0.2279315
```




```r
svm_perf = performance(svm_pred, "tpr", "fpr")

plot(lrr_perf)

lines(knn_perf@x.values[[1]], knn_perf@y.values[[1]], col = "green", lwd = 2.5)
lines(nb_perf@x.values[[1]], nb_perf@y.values[[1]], col = "red", lwd = 2.5)
lines(ada_perf@x.values[[1]], ada_perf@y.values[[1]], col = "blue", lwd = 2.5)
lines(dt_perf@x.values[[1]], dt_perf@y.values[[1]], col = "yellow", lwd = 2.5)
lines(lrr_perf@x.values[[1]], lrr_perf@y.values[[1]], col = "black", lwd = 2.5)
lines(svm_perf@x.values[[1]], svm_perf@y.values[[1]], col = "purple", lwd = 2.5)

legend(x = "topright", y = 5, legend = c("lr", "knn", "nb", "ada", "dt", "svm"),
    lty = c(1, 1, 1, 1), col = c("black", "green", "red", "blue", "yellow",
        "purple"), cex = 0.75)
```



4.Summarize the model performance based on the table and the ROC plot in one or two paragraphs.

Summary:

From the summary table we can see,decision tree model get the best performaonce on accuaracy. Both default tree and prune tree get the accuracy over 0.71, which is much higher than the other classification models. Adaboost get the accuracy over 0.69, which is the second hihgest score in the all of the classification techniques. The accuracy of loigistic regression, Naive Bayesian and SVM are almost in the same level, around 0.62 to 0.63. Knn shows the worst performance on accuracy, which is lower than 0.5. Decision tree and Adaboost also show good performance on precision. Both of them are over 0.5, while the other models' precision score are all lower than 0.5. Knn also shows the worst score on precision based on n=2 and n=3. Logistic reegression gives us the best score on recall. which is over 0.78. Naive Byesian and SVM show a middle score which are around 0.77. Knn still shows the worst score in recall, which is o.52. In F-score, Adaboost shows the best score and pruned decision tree shows the second best score and logistic regression comes after. The lowest score for f-score still on knn. For AUC, most of the models' score are around 0.85, only knn is around 0.4 which is the lowest score. Adaboost and logistic regression shows the highest score in AUC which is over 0.87. In general, the best model we can see from the summary table is Adaboost, SVM comes next and logistic regression comes the third. In SVM model, using kernel "sigmoid" will return the best AUC score. Desicion tree performs well on accuracy and precision and default tree performs better than pruned tree, Knn shows the worst performance on all score. When k=2, we will get the highet AUC in knn model.

According to barchart plot of f-score, we can see that, Adaboost has the highest score and all knn models show very low score. Decision tree has the second highest f-score and pruned tree performs better than default tree. Logistic regression model returns the score lower than decision tree but higher than SVM. SVM's score is better than Naive Bayesian while SVM with kernel "radical" shows the best score in all SVM model. Based on the barplot of AUC, we can get the conclusion that, Adaboost has the highest score and the second one is logistic regression. All SVM model can be seen as the third highest AUC here. Both desicion tree and Knn show bad performance on AUC. In ROC curve, a test with perfect discrimination has a ROC curve that passes through the upper left corner. Therefore the closer the ROC curve is to the upper left corner, the higher the overall accuracy of the test. From our ROC curve, we can know that, except knn, the other models all show a good performance on sensitivity and specificity. Most of the models show some extent of outfit. The purple and blue lines are much closer to the upper left corner, which they have better performance on sensitivity and specificity than the other models. Green line is far from the left upper corner which shows the worst performance. Therefore, Adaboost has the best performance and SVM in general has the second best performance. knn still has the worst performance and this model may not very suitable for this dataset. The conclusion from the graph is consistent with the conclusion from the summary table.