Algoritmos de Aprendizado de Máquina

Sobre a base de dados

O conjunto de dados utilizado é chamado "Adult" e foi derivado do banco de dados do censo dos EUA. Foi preparado por Barry Becker a partir do censo de 1994. O propósito desse conjunto de dados é prever se uma pessoa ganha mais ou menos de US$ 50.000 por ano.

Link do conjunto de dados: https://archive.ics.uci.edu/ml/datasets/Adult

Carregando os dados

```
train_data <- read.csv(file = "train_data.csv")
test_data <- read.csv(file = "test_data.csv")
```

Table 1: Adult Train Data - Part 1

age	workclass	education_num	marital_status	occupation	race	sex
39	3	13	2	0	4	0
50	2	13	0	3	4	0
38	1	9	1	5	4	0
53	1	7	0	5	2	0
28	1	13	0	9	2	1

Table 2: Adult Train Data - Part 2

capital_gain	capital_loss	hours_per_week	native_country	income
2174	0	40	0	0
0	0	13	0	0
0	0	40	0	0
0	0	40	0	0
0	0	40	0	0

Table 3: Adult Test Data - Part 1

age	workclass	education_num	marital_status	occupation	race	sex
38	1	9	0	4	4	0
28	3	12	0	10	4	0
44	1	10	0	6	2	0
18	1	10	2	9	4	0

age	workclass	education_num	marital_status	occupation	race	sex
34	1	6	2	7	4	1

Table 4: Adult Test Data - Part 2

capital_gain	capital_loss	hours_per_week	native_country	income
0	0	50	0	0
0	0	40	0	1
7688	0	40	0	1
0	0	30	0	0
0	0	30	0	0

1. Algoritmo Baseline (Classe Majoritária)

O algoritmo baseline é um algoritmo simples que serve como referência para avaliar a eficácia de outros algoritmos. O algoritmo baseline prevê a classe majoritária em todos os casos.

```
train_data$income_Baseline <- 0

true_negatives <- sum(train_data$income_Baseline == 0 & train_data$income == 0)
false_positives <- sum(train_data$income_Baseline == 0 & train_data$income == 1)
false_negatives <- 0
true_positives <- 0


cat("Matriz de Confusão:\n")
cat("           Previsto 0    Previsto 1\n")
cat(sprintf("Real 0:     %d            %d\n", true_negatives, false_positives))
cat(sprintf("Real 1:     %d            %d\n", false_negatives, true_positives))
```

```
## Matriz de Confusão:

##            Previsto 0    Previsto 1

## Real 0:     20873            7177

## Real 1:     0            0
```

```
accuracy_baseline <- (true_negatives + true_positives) / length(train_data$income)
cat("Acurácia:", accuracy_baseline, "\n")
```

```
## Acurácia: 0.7441355
```

```
if (true_positives + false_positives == 0) {
  cat("Precisão: Não definida (nenhuma previsão positiva)\n")
} else {
  precision_baseline <- true_positives / (true_positives + false_positives)
  cat("Precisão:", precision_baseline, "\n")
}
```

```
## Precisão: 0
```

```
#calcular recall
if (true_positives + false_negatives == 0) {
  cat("Recall: Não definido (nenhum valor real positivo)\n")
} else {
  recall_baseline <- true_positives / (true_positives + false_negatives)
  cat("Recall:", recall_baseline, "\n")
}
```

```
## Recall: Não definido (nenhum valor real positivo)
```

```
train_data$income_Baseline <- remove()
```

Com base nos resultados podemos ver que as métricas utilizadas foram, Acurácia, Precisão, Recall, e matriz de Confusão. A acurácia do algoritmo baseline é de 0.7441355. A precisão é de 0 e o recall é de 0. Isso ocorre porque o algoritmo baseline prevê a classe majoritária em todos os casos, ou seja, prevê que ninguém ganha mais de US$ 50.000 por ano.

2. Definição da técnica de Validação

Foi Utilizado a técnica de validação cruzada k-fold com k = 5. Pois a validação cruzada k-fold é uma técnica de validação de modelo que divide o conjunto de dados em k subconjuntos menores. O modelo é treinado em k-1 subconjuntos e testado no subconjunto restante. Esse processo é repetido k vezes, com cada subconjunto sendo usado como conjunto de teste uma vez. A validação cruzada k-fold é uma técnica eficaz para avaliar a capacidade de generalização de um modelo.

3. Algoritmo KNN

O algoritmo KNN (K-Nearest Neighbors) é um algoritmo de aprendizado supervisionado que pode ser usado para classificação e regressão. O algoritmo KNN classifica um novo ponto de dados com base nos pontos de dados existentes que estão mais próximos a ele. O algoritmo KNN tem um parâmetro chamado k, que é o número de vizinhos mais próximos a serem considerados.

```
if (!require(class)) install.packages("class")
```

```
## Carregando pacotes exigidos: class
```

```
library(class)

target <- train_data$income
K <- 5
folds <- cut(seq(1, nrow(train_data)), breaks=K, labels=FALSE)

accuracy_list <- c()
precision_list <- c()

# K-Fold Cross-Validation manual
for (i in 1:K) {
```

```r
  test_indexes <- which(folds == i, arr.ind = TRUE)
  teste <- train_data[test_indexes, ]
  treino <- train_data[-test_indexes, ]

  # Definir os atributos preditores e a variável target
  train_features <- treino[, -ncol(treino)]
  train_labels <- treino$income
  test_features <- teste[, -ncol(teste)]
  test_labels <- teste$income

  # Aplicar o algoritmo K-NN (usando K = 5 como exemplo)
  knn_pred <- knn(train = train_features, test = test_features, cl = train_labels, k = 5)


  accuracy <- sum(knn_pred == test_labels) / length(test_labels)
  accuracy_list <- c(accuracy_list, accuracy)

  true_positives <- sum(knn_pred == 1 & test_labels == 1)
  false_positives <- sum(knn_pred == 1 & test_labels == 0)
  if (true_positives + false_positives == 0) {
    precision <- 0
  } else {
    precision <- true_positives / (true_positives + false_positives)
    precision_list <- c(precision_list, precision)
  }

  #calcular recall
  false_negatives <- sum(knn_pred == 0 & test_labels == 1)
  if (true_positives + false_negatives == 0) {
    recall <- 0
  } else {
    recall <- true_positives / (true_positives + false_negatives)
  }


  cat("Matriz de Confusão para o fold", i, ":\n")
  cat("           Previsto 0   Previsto 1\n")
  cat("Real 0:    ", sum(knn_pred == 0 & test_labels == 0), "        ", sum(knn_pred == 1 & test_labels
  cat("Real 1:    ", sum(knn_pred == 0 & test_labels == 1), "        ", sum(knn_pred == 1 & test_labels
}
```

```
## Matriz de Confusão para o fold 1 :
##            Previsto 0   Previsto 1
## Real 0:     3849          362
## Real 1:     602          797
##
## Matriz de Confusão para o fold 2 :
##            Previsto 0   Previsto 1
## Real 0:     3849          380
## Real 1:     562          819
##
## Matriz de Confusão para o fold 3 :
##            Previsto 0   Previsto 1
```

```
## Real 0:     3794          368
## Real 1:     590          858
##
## Matriz de Confusão para o fold 4 :
##            Previsto 0   Previsto 1
## Real 0:     3790          378
## Real 1:     577          865
##
## Matriz de Confusão para o fold 5 :
##            Previsto 0   Previsto 1
## Real 0:     3731          372
## Real 1:     602          905
```

```r
# Média das acurácias dos K folds
mean_accuracy_knn <- mean(accuracy_list)
print(paste("Acurácia média da validação cruzada:", mean_accuracy_knn))
```

```
## [1] "Acurácia média da validação cruzada: 0.829126559714795"
```

```r
# Média das precisões dos K folds
mean_precision_knn <- mean(precision_list)
print(paste("Precisão média da validação cruzada:", mean_precision_knn))
```

```
## [1] "Precisão média da validação cruzada: 0.695031428079305"
```

```r
# Média dos recalls dos K folds
mean_recall_knn <- mean(recall)
print(paste("Recall médio da validação cruzada:", mean_recall_knn))
```

```
## [1] "Recall médio da validação cruzada: 0.600530856005309"
```

4. Árvore de Decisão

O algoritmo de árvore de decisão é um algoritmo de aprendizado supervisionado que pode ser usado para classificação e regressão. O algoritmo de árvore de decisão divide o conjunto de dados em subconjuntos menores com base em um conjunto de regras. O algoritmo de árvore de decisão tem parâmetros que controlam a profundidade da árvore e o número mínimo de amostras necessárias para dividir um nó.

```r
if (!require(rpart)) install.packages("rpart")
```

```
## Carregando pacotes exigidos: rpart
```

```r
library(rpart)

# Inicializar variáveis para guardar resultados
accuracy_list <- c()
precision_list <- c()

for (i in 1:K) {
```

```r
  # Separar os dados em treinamento e teste
  test_indexes <- which(folds == i, arr.ind = TRUE)
  teste <- train_data[test_indexes, ]
  treino <- train_data[-test_indexes, ]

  # Definir os atributos preditores e a variável target
  train_features <- treino[, -ncol(treino)]
  train_labels <- treino$income
  test_features <- teste[, -ncol(teste)]
  test_labels <- teste$income

  # Aplicar o algoritmo Árvore de Decisão
  decision_tree <- rpart(income ~ ., data = treino, method = "class")
  rpart_pred <- predict(decision_tree, test_features, type = "class")

  accuracy <- sum(rpart_pred == test_labels) / length(test_labels)
  accuracy_list <- c(accuracy_list, accuracy)

  true_positives <- sum(rpart_pred == 1 & test_labels == 1)
  false_positives <- sum(rpart_pred == 1 & test_labels == 0)
  if (true_positives + false_positives == 0) {
    precision <- 0
  } else {
    precision <- true_positives / (true_positives + false_positives)
    precision_list <- c(precision_list, precision)
  }

  false_negatives <- sum(rpart_pred == 0 & test_labels == 1)
  if (true_positives + false_negatives == 0) {
    recall <- 0
  } else {
    recall <- true_positives / (true_positives + false_negatives)
  }

  cat("Matriz de Confusão para o fold", i, ":\n")
  cat("           Previsto 0   Previsto 1\n")
  cat("Real 0:    ", sum(rpart_pred == 0 & test_labels == 0), "        ", sum(rpart_pred == 1 & test_lab
  cat("Real 1:    ", sum(rpart_pred == 0 & test_labels == 1), "        ", sum(rpart_pred == 1 & test_lab
}
```

```
## Matriz de Confusão para o fold 1 :
##            Previsto 0   Previsto 1
## Real 0:     3937          274
## Real 1:     659          740
##
## Matriz de Confusão para o fold 2 :
##            Previsto 0   Previsto 1
## Real 0:     3942          287
## Real 1:     633          748
##
## Matriz de Confusão para o fold 3 :
##            Previsto 0   Previsto 1
## Real 0:     3894          268
```

```
## Real 1:     622          826
##
## Matriz de Confusão para o fold 4 :
##            Previsto 0   Previsto 1
## Real 0:     3851          317
## Real 1:     614          828
##
## Matriz de Confusão para o fold 5 :
##            Previsto 0   Previsto 1
## Real 0:     3828          275
## Real 1:     655          852
```

```r
# Média das acurácias dos K folds
mean_accuracy_DecTree <- mean(accuracy_list)
print(paste("Acurácia média da validação cruzada:", mean_accuracy_DecTree))
```

```
## [1] "Acurácia média da validação cruzada: 0.835864527629234"
```

```r
# Média das precisões dos K folds
mean_precision_DecTree <- mean(precision_list)
print(paste("Precisão média da validação cruzada:", mean_precision_DecTree))
```

```
## [1] "Precisão média da validação cruzada: 0.737329846170924"
```

```r
# Média dos recalls dos K folds
mean_recall_DecTree <- mean(recall)
print(paste("Recall médio da validação cruzada:", mean_recall_DecTree))
```

```
## [1] "Recall médio da validação cruzada: 0.565361645653616"
```

5. Redes Neurais

As redes neurais são um tipo de algoritmo de aprendizado de máquina que são inspirados no funcionamento do cérebro humano. As redes neurais são compostas por camadas de neurônios que são conectados entre si. Cada neurônio recebe entradas, realiza um cálculo e passa a saída para os neurônios da próxima camada. As redes neurais têm parâmetros que controlam o número de camadas, o número de neurônios em cada camada e a função de ativação.

```r
if (!require(nnet)) install.packages("nnet")
```

```
## Carregando pacotes exigidos: nnet
```

```r
library(nnet)
train_data$income <- as.factor(train_data$income)

accuracy_list <- c()
precision_list <- c()

for (i in 1:K) {
```

```r
  # Separar os dados em treinamento e teste
  test_indexes <- which(folds == i, arr.ind = TRUE)
  teste <- train_data[test_indexes, ]
  treino <- train_data[-test_indexes, ]

  # Definir os atributos preditores e a variável target
  train_features <- treino[, -ncol(treino)]
  train_labels <- treino$income
  test_features <- teste[, -ncol(teste)]
  test_labels <- teste$income

  # Aplicar o algoritmo Rede Neural
  mlp_model <- nnet(income ~ ., data = treino, size = 5, maxit = 1000)
  mlp_pred <- predict(mlp_model, test_features, type = "class")


  accuracy <- sum(mlp_pred == test_labels) / length(test_labels)
  accuracy_list <- c(accuracy_list, accuracy)
  true_positives <- sum(mlp_pred == 1 & test_labels == 1)
  false_positives <- sum(mlp_pred == 1 & test_labels == 0)
  if (true_positives + false_positives == 0) {
    precision <- 0
  } else {
    precision <- true_positives / (true_positives + false_positives)
    precision_list <- c(precision_list, precision)
  }


  false_negatives <- sum(mlp_pred == 0 & test_labels == 1)
  if (true_positives + false_negatives == 0) {
    recall <- 0
  } else {
    recall <- true_positives / (true_positives + false_negatives)
  }

  cat("Matriz de Confusão para o fold", i, ":\n")
  cat("           Previsto 0   Previsto 1\n")
  cat("Real 0:    ", sum(mlp_pred == 0 & test_labels == 0), "        ", sum(mlp_pred == 1 & test_labels
  cat("Real 1:    ", sum(mlp_pred == 0 & test_labels == 1), "        ", sum(mlp_pred == 1 & test_labels
}
```

```
## # weights:  66
## initial  value 12939.788720
## iter  10 value 11804.507910
## iter  20 value 11117.409141
## iter  30 value 11001.519792
## iter  40 value 10916.296678
## iter  50 value 10814.251306
## iter  60 value 10290.004661
## iter  70 value 9147.389526
## iter  80 value 8680.754338
## iter  90 value 8457.900968
## iter 100 value 8327.882479
```

```
## iter 110 value 8256.261477
## iter 120 value 8187.631466
## iter 130 value 8123.364526
## iter 140 value 7996.697216
## iter 150 value 7863.332701
## iter 160 value 7795.455580
## iter 170 value 7773.567714
## iter 180 value 7766.517483
## iter 190 value 7736.277915
## iter 200 value 7729.804185
## iter 210 value 7727.620885
## iter 220 value 7713.207167
## iter 230 value 7694.891608
## iter 240 value 7686.742093
## iter 250 value 7684.398985
## iter 260 value 7683.282283
## iter 270 value 7679.004060
## iter 280 value 7676.973114
## iter 290 value 7675.269287
## iter 300 value 7670.931893
## iter 310 value 7667.770625
## iter 320 value 7667.214929
## iter 330 value 7666.794444
## iter 340 value 7666.519633
## iter 350 value 7662.074340
## iter 360 value 7661.398804
## iter 370 value 7660.653566
## iter 380 value 7659.595702
## iter 390 value 7649.733225
## iter 400 value 7632.651558
## iter 410 value 7630.536229
## iter 420 value 7628.565063
## iter 430 value 7628.432357
## iter 440 value 7627.654731
## iter 450 value 7621.496141
## iter 460 value 7619.338946
## iter 470 value 7616.725452
## iter 480 value 7616.303428
## iter 490 value 7616.251320
## iter 500 value 7616.213588
## iter 500 value 7616.213565
## iter 500 value 7616.213565
## final  value 7616.213565
## converged
## Matriz de Confusão para o fold 1 :
##            Previsto 0   Previsto 1
## Real 0:     3941          270
## Real 1:     654          745
##
## # weights:  66
## initial  value 19397.939361
## iter  10 value 12247.789531
## iter  20 value 11270.218676
## iter  30 value 11144.635162
```

```
## iter  40 value 11068.129827
## iter  50 value 10986.335348
## iter  60 value 10927.839148
## iter  70 value 10545.475047
## iter  80 value 9928.316460
## iter  90 value 9258.300310
## iter 100 value 8246.450239
## iter 110 value 8052.822310
## iter 120 value 7988.432025
## iter 130 value 7954.651192
## iter 140 value 7946.994576
## iter 150 value 7931.844189
## iter 160 value 7929.470323
## iter 170 value 7929.196201
## iter 180 value 7927.015739
## iter 190 value 7922.824716
## iter 200 value 7921.170126
## iter 210 value 7920.605670
## iter 220 value 7920.410430
## iter 230 value 7920.088698
## iter 230 value 7920.088658
## iter 230 value 7920.088657
## final  value 7920.088657
## converged
## Matriz de Confusão para o fold 2 :
##            Previsto 0   Previsto 1
## Real 0:     3902          327
## Real 1:     607          774
##
## # weights:  66
## initial  value 16568.432723
## iter  10 value 11823.480765
## iter  20 value 11404.143744
## iter  30 value 11129.071841
## iter  40 value 10976.846593
## iter  50 value 10949.985412
## iter  60 value 10917.349482
## iter  70 value 10503.456802
## iter  80 value 9764.164801
## iter  90 value 9083.844066
## iter 100 value 8597.472811
## iter 110 value 8481.372883
## iter 120 value 8347.174055
## iter 130 value 8322.190055
## iter 140 value 8263.926561
## iter 150 value 8176.578884
## iter 160 value 8138.742934
## iter 170 value 8097.844732
## iter 180 value 8026.077060
## iter 190 value 7990.496061
## iter 200 value 7921.942048
## iter 210 value 7893.793938
## iter 220 value 7841.884766
## iter 230 value 7816.843827
```

```
## iter 240 value 7768.733692
## iter 250 value 7747.015184
## iter 260 value 7718.503786
## iter 270 value 7710.211498
## iter 280 value 7700.712993
## iter 290 value 7697.434825
## iter 300 value 7696.916958
## iter 310 value 7696.826050
## iter 320 value 7696.662650
## iter 330 value 7696.539071
## iter 340 value 7696.374782
## iter 350 value 7696.180701
## iter 360 value 7695.185390
## iter 370 value 7694.234266
## final  value 7694.198244
## converged
## Matriz de Confusão para o fold 3 :
##            Previsto 0   Previsto 1
## Real 0:     3826          336
## Real 1:     591          857
##
## # weights:  66
## initial  value 12047.675773
## iter  10 value 11771.486391
## iter  20 value 11636.917736
## iter  30 value 10950.191976
## iter  40 value 10204.617683
## iter  50 value 9671.854479
## iter  60 value 9581.873365
## iter  70 value 9549.148357
## iter  80 value 9467.988144
## iter  90 value 9394.762024
## iter 100 value 9275.824237
## iter 110 value 9043.789791
## iter 120 value 8947.698325
## iter 130 value 8912.562230
## iter 140 value 8904.731631
## iter 150 value 8899.822405
## iter 160 value 8888.952193
## iter 170 value 8885.538407
## iter 180 value 8865.923162
## iter 190 value 8861.530156
## iter 200 value 8861.330873
## iter 210 value 8861.297520
## final  value 8861.287043
## converged
## Matriz de Confusão para o fold 4 :
##            Previsto 0   Previsto 1
## Real 0:     3540          628
## Real 1:     551          891
##
## # weights:  66
## initial  value 23186.398464
## iter  10 value 12005.988042
```

```
## iter  20 value 11932.653921
## iter  30 value 11721.325179
## iter  40 value 11023.893210
## iter  50 value 10017.570020
## iter  60 value 9034.266938
## iter  70 value 8832.462347
## iter  80 value 8651.502703
## iter  90 value 8601.971051
## iter 100 value 8597.327940
## iter 110 value 8565.839628
## iter 120 value 8531.525108
## iter 130 value 8511.735979
## iter 140 value 8494.182266
## iter 150 value 8441.660259
## iter 160 value 8410.806039
## iter 170 value 8396.009782
## iter 180 value 8247.115225
## iter 190 value 8109.922065
## iter 200 value 8092.820612
## iter 210 value 8089.585954
## final  value 8089.445122
## converged
## Matriz de Confusão para o fold 5 :
##            Previsto 0   Previsto 1
## Real 0:     3824          279
## Real 1:     778          729
```

```
mean_accuracy_mlp <- mean(accuracy_list)
print(paste("Acurácia média da validação cruzada:", mean_accuracy_mlp))
```

```
## [1] "Acurácia média da validação cruzada: 0.820998217468806"
```

```
mean_precision_mlp <- mean(precision_list)
print(paste("Precisão média da validação cruzada:", mean_precision_mlp))
```

```
## [1] "Precisão média da validação cruzada: 0.693025780720341"
```

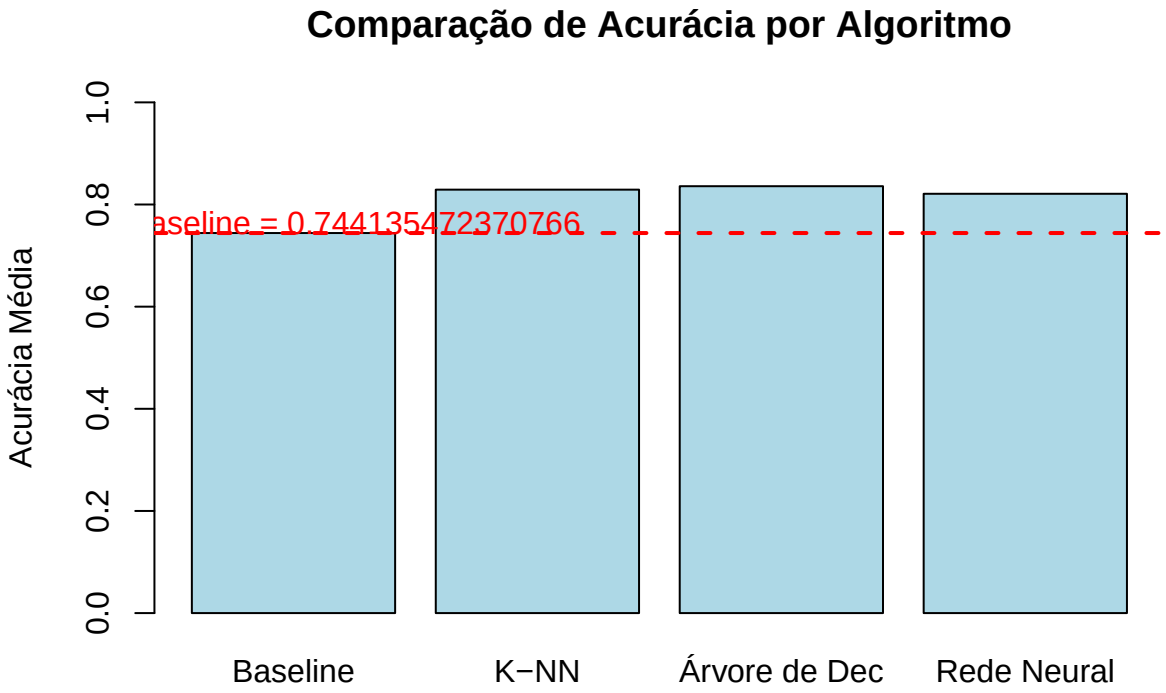
```
mean_recall_mlp <- mean(recall)
print(paste("Recall médio da validação cruzada:", mean_recall_mlp))
```

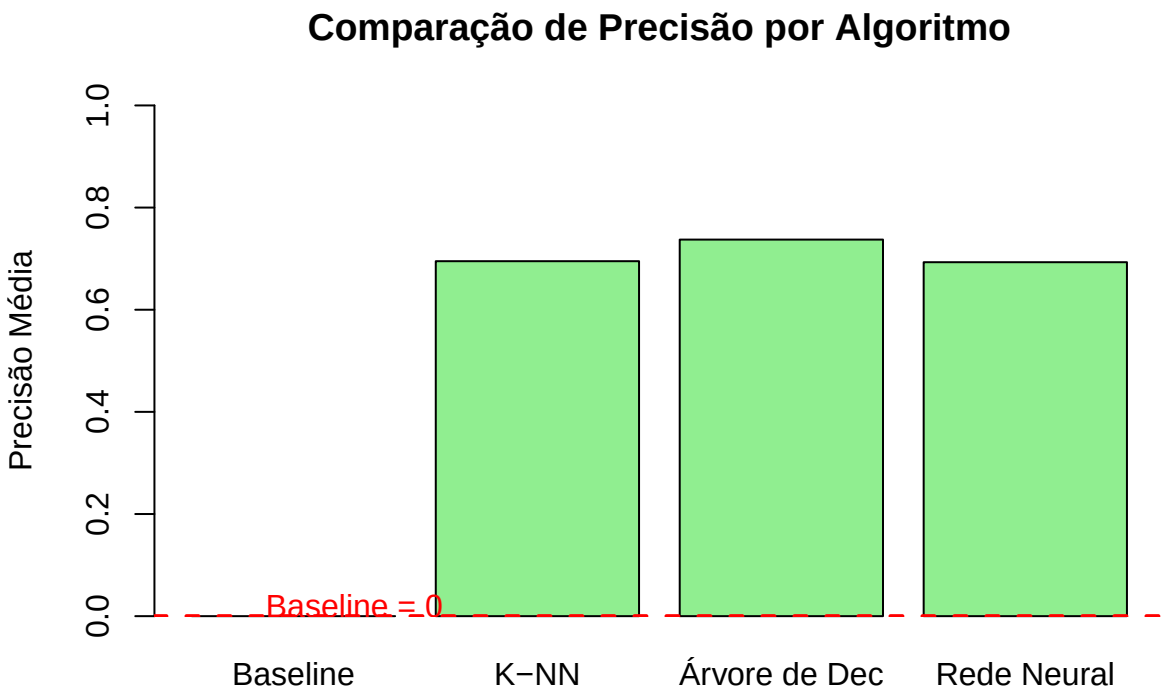
```
## [1] "Recall médio da validação cruzada: 0.483742534837425"
```

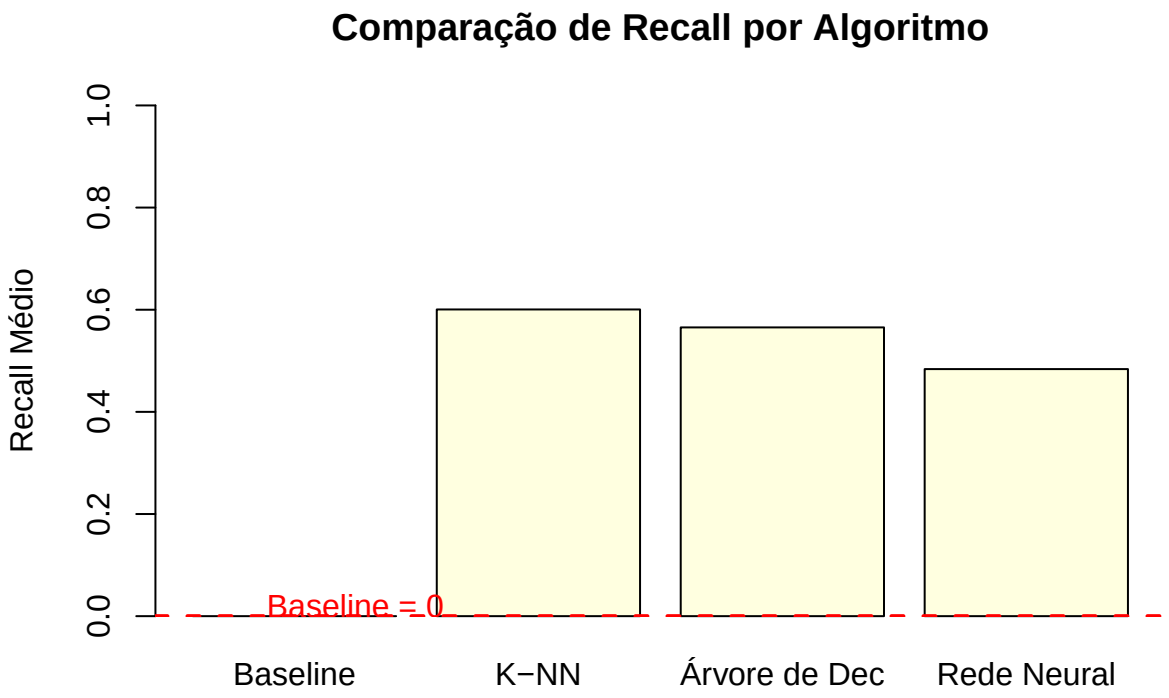
6. Comparação dos Algoritmos

```
## Carregando pacotes exigidos: kableExtra
```

	Resultados dos Algoritmos		
Algoritmo	**Acurácia**	**Precisão**	**Recall**
Baseline	**0.7441**	**0.0000**	**0.0000**
K-NN	**0.8291**	**0.6950**	**0.6005**
Árvore de Decisão	**0.8359**	**0.7373**	**0.5654**
Rede Neural (MLP)	**0.8210**	**0.6930**	**0.4837**




Comparação de Precisão por Algoritmo
Precisão Média
0.0
0.2
0.4
0.6
0.8
1.0
Baseline = 0
Baseline
K−NN
Árvore de Dec
Rede Neural



Conclusão

As métricas importantes nesse contexto são Precisão e Recall pois, a Precisão é importante para evitar falsos positivos(ou seja, prever corretamente as pessoas que ganham mais de 50.000). E Recall é importante para evitar falsos negativos (ou seja, garantir que todas as pessoas que ganham mais de 50.000 sejam corretamente identificadas).

Aplicar o algoritmo Árvore de Decisão na base de teste

```
decision_tree <- rpart(income ~ ., data = train_data, method = "class")
rpart_pred <- predict(decision_tree, test_features, type = "class")

true_negatives <- sum(rpart_pred == 0 & test_labels == 0)
false_positives <- sum(rpart_pred == 1 & test_labels == 0)
false_negatives <- sum(rpart_pred == 0 & test_labels == 1)
true_positives <- sum(rpart_pred == 1 & test_labels == 1)

cat("Matriz de Confusão para a base de teste:\n")
```

```
## Matriz de Confusão para a base de teste:
```

```r
cat("           Previsto 0   Previsto 1\n")
```

```
##            Previsto 0   Previsto 1
```

```r
cat("Real 0:    ", true_negatives, "        ", false_positives, "\n")
```

```
## Real 0:     3828          275
```

```r
cat("Real 1:    ", false_negatives, "        ", true_positives, "\n\n")
```

```
## Real 1:     655          852
```

```r
accuracy_rpart_test <- (true_negatives + true_positives) / length(test_labels)
cat("Acurácia na base de teste:", accuracy_rpart_test, "\n")
```

```
## Acurácia na base de teste: 0.8342246
```

```r
if (true_positives + false_positives == 0) {
  cat("Precisão na base de teste: Não definida (nenhuma previsão positiva)\n")
} else {
  precision_rpart_test <- true_positives / (true_positives + false_positives)
  cat("Precisão na base de teste:", precision_rpart_test, "\n")
}
```

```
## Precisão na base de teste: 0.7559894
```

```r
recall <- true_positives / (true_positives + false_negatives)
cat("Recall na base de teste:", recall, "\n")
```

```
## Recall na base de teste: 0.5653616
```