RELATÓRIO DA DISCIPLINA RESOLUÇÃO DE PROBLEMAS VIA MODELAGEM MATEMÁTICA UNIFESP

Projeto – Identificação de fraudes em cartão de crédito

Amanda Gomes Vetorazzi
Danielle Guanaes
Thiago Siqueira Santos

i. Proponente – Mateus Vendramini Polizeli, orientando do Professor Luiz Felipe Bueno e funcionário do departamento de fraudes do Itaú.

ii. Definição do problema e objetivo:

De acordo com o jornal Estadão, entre janeiro e agosto de 2018, foram registrados 3,6 fraudes por minuto em cartões de crédito no Brasil. Esses dados levam nosso país a segunda posição mundial, ao lado dos EUA, em incidências desse tipo de fraude.

O centro do nosso problema se baseia em notícias como essa. Como podemos mitigar esse número de fraudes que, ao mesmo tempo, trazem prejuízo para instituições financeiras e transtorno para o consumidor? De quais formas podemos utilizar algoritmos e processos já conhecidos para identificar uma fraude antes que ela aconteça? Que tipo de informações um gestor da área de fraudes bancárias precisaria, na impossibilidade de prever todas as transações falsas, para calibrar um sistema que não permitisse que um número excessivo de bloqueios de operações honestas atrapalhasse a imagem da instituição?

Com esses questionamentos como norteadores, nos foi indicado um conjunto de dados que nos permitisse aplicar algoritmos de *machine laearning* para identificarmos quais transações eram falsas.

O conjunto de dados (*Credit Card Fraud Detection*) em questão é público e está disponível no *Kaggle (https://www.kaggle.com/mlg-ulb/creditcardfraud)*. Ele contém transações ocorridas em dois dias de setembro de 2013 e feitas por possuidores de cartões de crédito europeus. Essa instância possui 30 (trinta) variáveis de entrada e uma variável de resposta (Class) que indica se determinada operação é fraudulenta. As variáveis V1 a V28 são componentes principais obtidas pela transformação PCA (Análise de Componentes Principais), tendo em vista a necessidade de confidencialidade. As únicas variáveis que não foram transformados com PCA foram:

- Tempo: tempo (segundos) entre cada transação, e a primeira do conjunto de dados;
- Quantia: quantia envolvida na da transação, que pode ser usada para um aprendizado sensível ao custo.

O sumário da descrição do conjunto de dados encontra-se na Tabela 1. Nota-se que o conjunto de dados está desbalanceado, isto é, a classe minoritária é muito menor do que a majoritária, o que necessariamente deve ser levado em consideração na resolução do problema.

Tabela 1. Sumário da descrição do conjunto de dados

Conjunto de dados	#Exemplos	#Atributos (simb/num)	%Ausentes	%Minoritária
Credit Card Fraud Detection	284.807	(0/30)	0	0,172

Em suma, o objetivo é, dado um conjunto de dados com centenas de milhares de transações de um sistema de cartão de crédito, detectar aquelas que podem fraudulentas, aplicar métricas para avaliar os resultados encontrados e estabelecer uma estratégia que guie um gestor para uma tomada de decisão.

iii. Fundamentos Teóricos

Em linhas gerais, problemas dessa natureza passam por um processo KDD para identificação de padrões compreensíveis, válidos, novos, potencialmente úteis, por meio da análise de um grande volume de dados. O processo consiste na aplicação de técnicas que objetivam a transformação dos dados armazenados em conhecimento útil para assim auxiliar a tomada de decisões assertivas. Suas etapas são a seleção dos dados, pré-processamento, transformação, mineração de dados através da utilização de métodos para descoberta de padrões e avaliação dos resultados. [Fayyad et al. 1996].

Dependendo dos métodos a serem utilizados na etapa de mineração dos dados, são necessárias transformações no conjunto de dados ou não ou não.

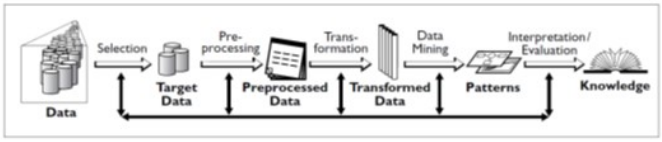

Figura 1. Etapas do processo de KDD [Fayyad et al, 1996]

Por se tratar de uma instância de teste, o *Credit Card Fraud Detection* do *Kaggle* já nos é apresentado na forma pré-processada. Assim, nossa preocupação antes de aplicarmos os algoritmos de *machine learnig* foi apenas no sentido de compreender melhor os dados ali presentes, procurando por dados faltantes, outliers, distribuição entre as classes e correlação entre as variáveis. Além disso, foi preciso subdividir o conjunto original em duas partes, uma destinada ao processo de treinamento e outra para os testes.

Os métodos que foram escolhidos para abordar o problema foram aqueles que a literatura apontava como potencialmente bons para dados desbalanceados, as árvores de decisão e as florestas aleatórias, e um método que fosse bem aplicado no processo de separação de dados, as redes neurais. Como o conjunto de dados a serem trabalhados era relativamente grande, todos os resultados apresentados neste relatório foram obtidos em linguagem R, utilizando pacotes já implementados.

Árvores de decisão é método no qual o aprendizado baseia-se em "dividir para conquistar" conhecido como particionamento recursivo. Baseia-se no aumento do ganho de informação, isto é, na redução da aleatoriedade, para selecionar as melhores variáveis de entrada para o particionamento em cada etapa. Existem vários algoritmos que utilizam esse método: Algoritmo de Hunt, CART, C4.5. Como resultado temos uma estrutura simbólica que representa o processo de decisão.

Neste trabalho utilizamos o algoritmo CART, que foi proposto por Breiman et al em 1984. No começo o nó raiz é dividido em dois filhos, cada filho se divide em netos e assim sucessivamente. Assim as árvores crescem até um tamanho máximo sem o uso de uma regra de parada. A árvore de maior tamanho é podada através da utilização do método de "cost-complexity pruning", na qual a poda é realizada no tamanho que menos contribui para a performance total da arvore.

Já para o algoritmo Random Forest, produz uma série de árvores de decisão diferentes utilizando subconjuntos do treinamento obtidos por processos aleatórios com o intuito de reduzir a variância. O número de árvores é um parâmetro de entrada do método e irá ser determinante no erro medido pelo processo. Quanto maior esse número, mais precisos serão os resultados obtidos e mais elevado será seu custo computacional.

Ao final do processo que gera as árvores da floresta, temos que o processo de predição é feito por uma média dos resultados individuais encontrados. É possível estabelecer uma classificação binária, dada pelo resultado indicado pela maioria das árvores, mas este item foi desconsiderado nas discussões que foram desenvolvidas neste trabalho.

Segundo Haykin, a teoria de redes neurais é inspirada na capacidade de reconhecimento de padrões do cérebro e possui elementos como neurônios, sinais, taxas de aprendizagem e sinapses. Uma rede neural é capaz de armazenar informações experimentais e deixá-las disponíveis para uso posterior.

Dentre as diversas formatações de redes, a multilayer perceptron é uma rede neural que possui três principais características. A função ativação, que deve ser uma função diferenciável; pelo menos um neurônio em pelo menos uma camada intermediária; e conexão entre os neurônios e essas camadas.

O treinamento de uma rede neural pelo algoritmo retro-propagação é feito em duas etapas. Na primeira etapa os sinais são absorvidos pela camada de entrada e transmitidos para as camadas intermediárias através de sinapses. Na segunda etapa, é produzido um erro, que compara os valores na camada de saída com os valores desejados. Assim, um processo de otimização é estabelecido com o intuito de

minimizar esse erro observado.

iv. Modelo

Após uma análise inicial dos dados, foi feita uma separação aleatória na proporção de 70%/30% da base original, criando assim um conjunto de treinamento e um conjunto de teste respectivamente.

O algoritmo *CART* foi utilizado para construir as árvores de decisão através do pacote *rpart* em R. Inicialmente fizemos a árvore completa e posteriormente foi feita uma poda, conforme o método de pós-poda. Esse procedimento reduz o tamanho da arvore sem reduzir a acurácia e evita o problema de *overfitting.*

O pacote rpart calcula o "*cost complexity pruning*" (cp) para cada nó folha da arvore e relaciona o mesmo com a erro da validação cruzada. A poda tenta estimar o valor do cp que garanta a melhor relação entre a acurácia e o tamanho da arvore. Através do gráfico do cp (*cost-complexity parameter*) selecionamos o tamanho ótimo da arvore. Uma boa escolha de cp para a poda é, geralmente, o valor mais à esquerda abaixo da linha horizontal, como podemos ver na Figura 2.

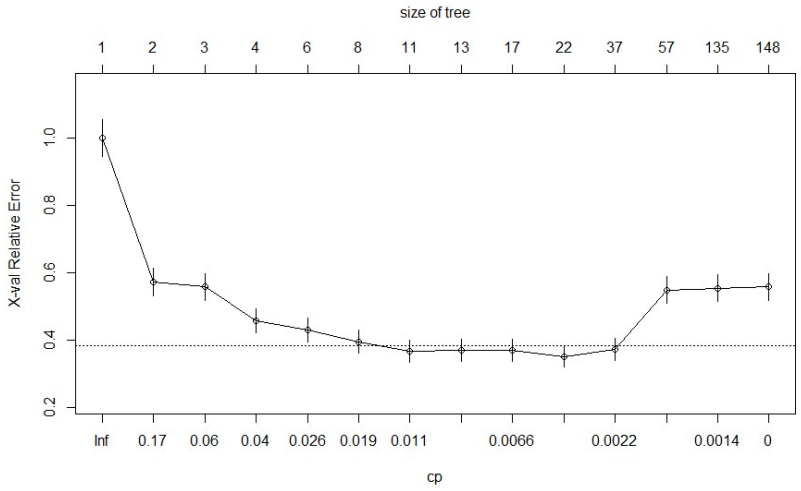

Figura 2. Gráfico de cp e tamanho da árvore utilizando validação cruzada

Implementado no pacote *randomForest* em R, inicialmente para 100 árvores e gradativamente aumentando este número, foi possível perceber que o erro estava estabilizando por volta de 200 árvores. Assim, de forma a garantir um resultado mais robusto, trabalhamos com um número 10 vezes maior, fixando nossos testes com 2000 árvores.

Com o auxílio do comando tuneRF, foi possível estabelecer qual seria o melhor valor para o parâmetro mtry, que estabelece qual é o tamanho dos subconjuntos que são gerados aleatoriamente para cada nova árvore. Como pode ser visto na Figura 3, o valor escolhido é aquele que minimiza o erro.

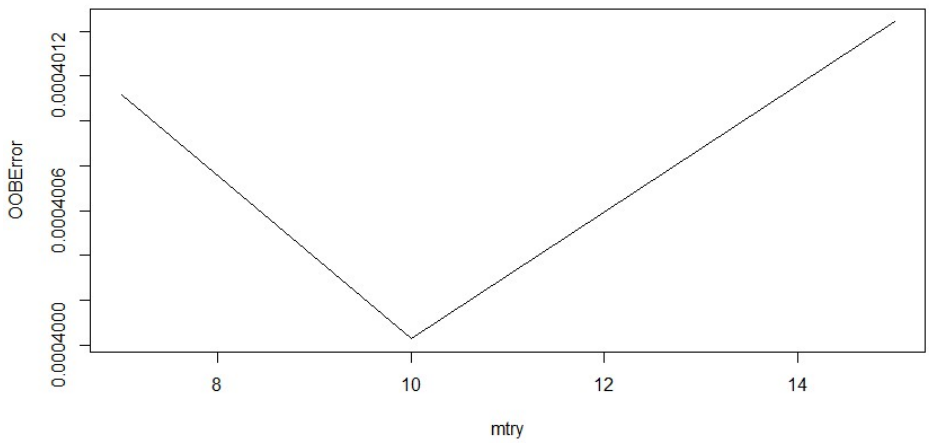

Figura 3. Gráfico que indica o melhor valor para o parâmetro mtry.

O código para implementação da rede neural se inicializa com o a normalização da base de dados para que o pacote *neuralnet* do R funcione corretamente. Como encontramos uma série de dificuldades para trabalhar com a totalidade do conjunto de treinamento utilizado nos métodos anteriores, construímos um subconjunto do mesmo, selecionando aleatoriamente 200 observações de fraudes e 400 honestas.

Como parâmetros de entrada, o pacote recebe o conjunto de treino, as quantidades de camada intermediárias e neurônios, um limitante para o erro, a quantidade de repetições e especificações sobre o cálculo do erro.

v. Resultados

v.1. Pré-processamento

Após uma análise inicial dos dados originais, podemos concluir pelos histogramas apresentados nas Figuras 4 e 5 que as distribuições não são idênticas quando comparamos a totalidade dos dados e os dados classificados como fraudes.

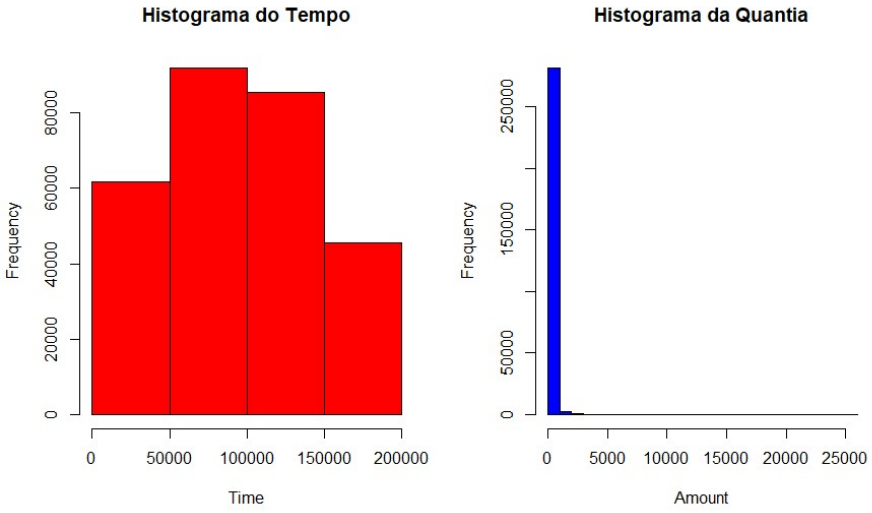

Figura 4. Histogramas para todas as operações

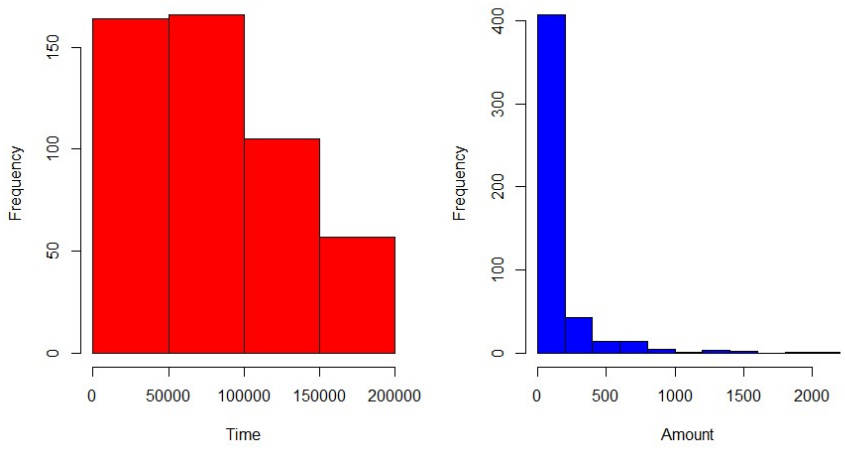

Figura 5. Histogramas para as operações fraudulentas

Partindo dos *boxplots* das variáveis, representados nas Figuras 6 a 9, verificamos que as medianas são próximas para a maioria dos casos e que há *outliers* no conjunto de dados com class igual a 1 e 0.

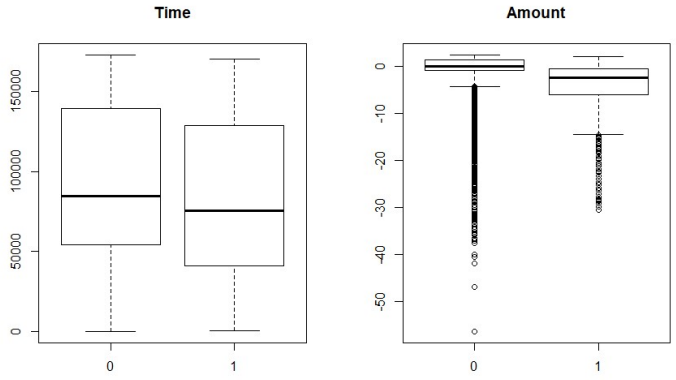

Figura 6. Bloxplot do tempo e da quantia

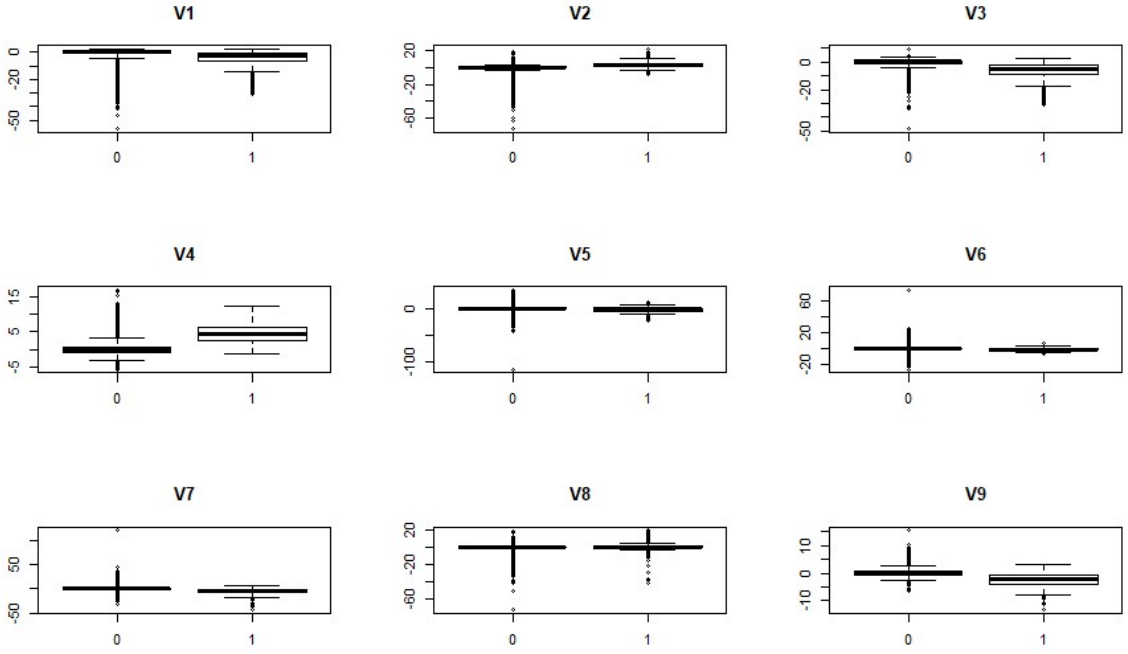

Figura 7. Bloxplot das variáveis V1 a V9

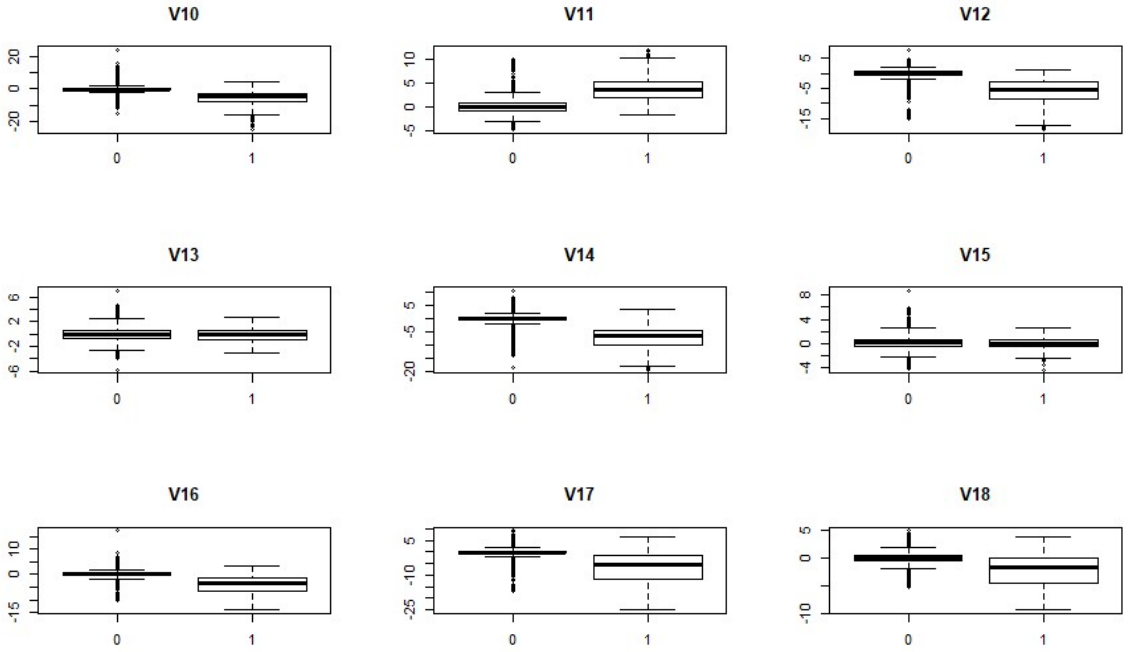

Figura 8. Bloxplot das variáveis V10 a V18

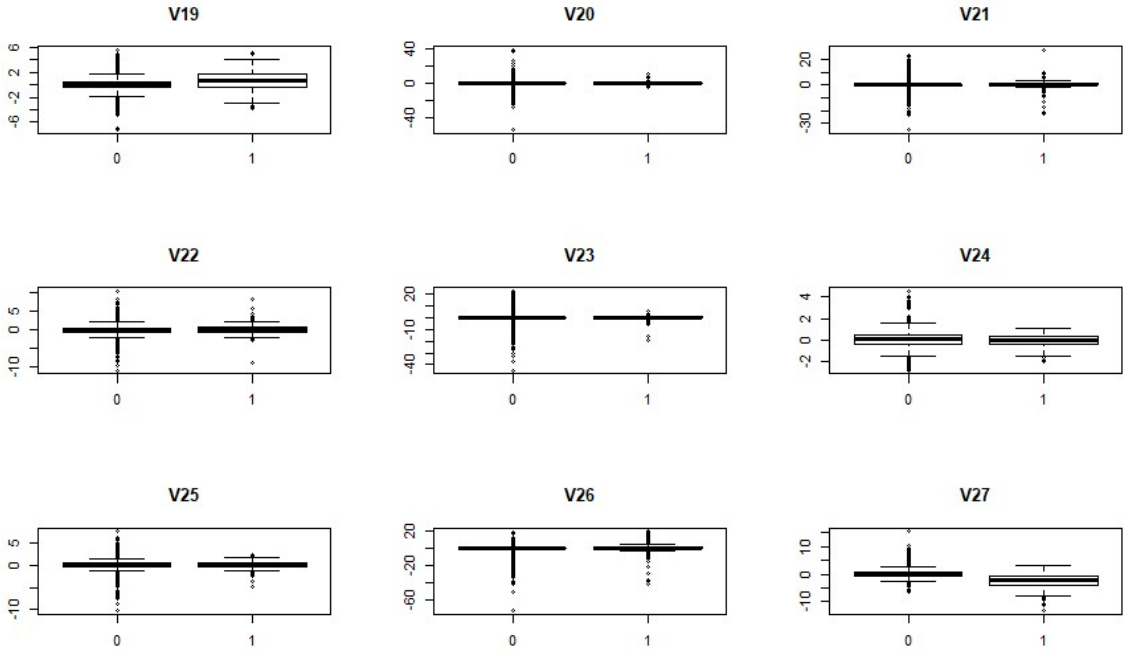

Figura 9. Bloxplot das variáveis V19 a V27

Por último, temos um gráfico na Figura 10 que apresenta a correlação entre as variáveis. É possível intuir que o processo de transformação PCA tenha minimizado esse parâmetro, baseando-se na observação que as variáveis que mantém a característica original, *time* e *amount*, são aquelas tem correlação mais elevada.

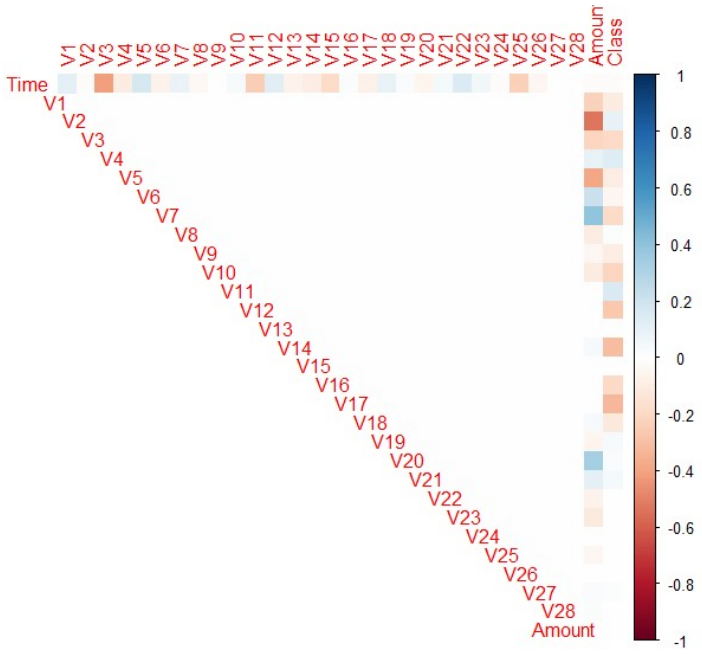

Figura 10. Correlação entre as variáveis

v.2. CART

Apesar de haver *outliers*, de os dados serem desbalanceados de haver correlação entre alguns atributos, não foi feito nenhuma transformação no conjunto de dados para utilização do algoritmo CART.

Os resultados do *rpart* para a árvore completa indicaram que poderia haver *overplotting*. Na Figura 2 constatamos que a arvore deve ser podada para um cp igual a 0,011. A árvore podada com cp igual a 0,011 consta na Figura 10.

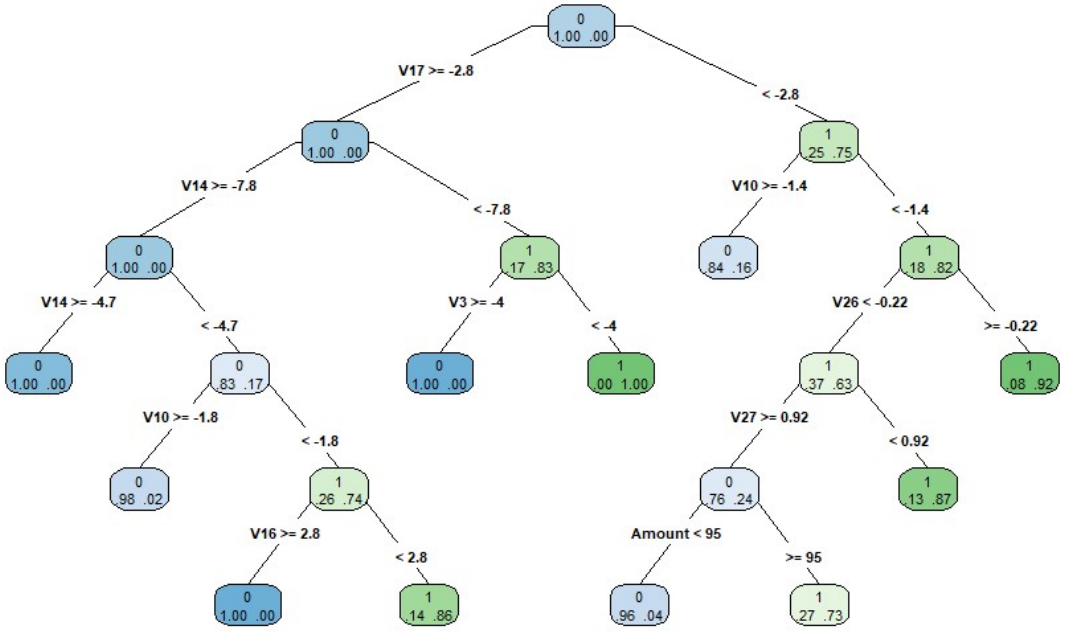

Figura 10. Árvore de decisão gerada pelo CART

Ao compararmos os resultados da AUPRC (Area under precision-recall curve), métrica no intervalo [0,1] comumente utilizada para avaliar métodos que classificam dados desbalanceados, tanto para o conjunto de dados de treino quanto para os de teste na árvore podada, observamos que os valores são compatíveis, indicando não

haver *overfitting*.

Tabela 2. Resultados da AUPRC para CART

Conjunto de dados	AUPRC
Treino	0,7575
Teste	0,7575

Na Figura 11 temos a importância dos atributos segundo o CART. Observamos que as variáveis mais importantes são V17, V12, V10, V16, V18 e V11.

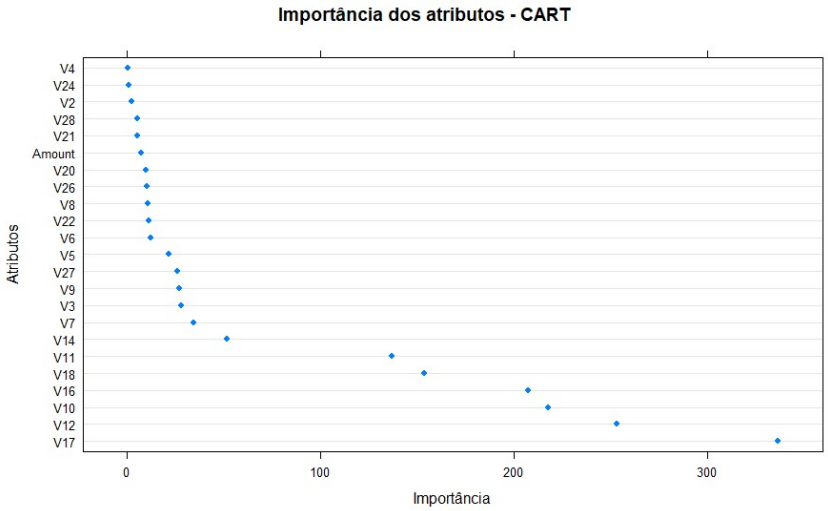

Figura 11. Gráfico da importância dos atributos gerado pelo CART

v.3. Random Forest

Os resultados encontrados pela Random forest, medidos novamente pela AUCPR foram, como era o esperado, melhores do que aqueles encontrados pelo CART, mesmo para um número pequeno de árvores, como pode ser visto na Tabela 3.

Tabela 3. Resultados da AUPRC para Random Forest

Número de árvores	AUPRC
100	0,867
200	0,884
2000	0,893

O algoritmo também apontou as variáveis de maior importância na decisão, Figura 12, e os resultados são bastante semelhantes aos encontrados pelo CART.

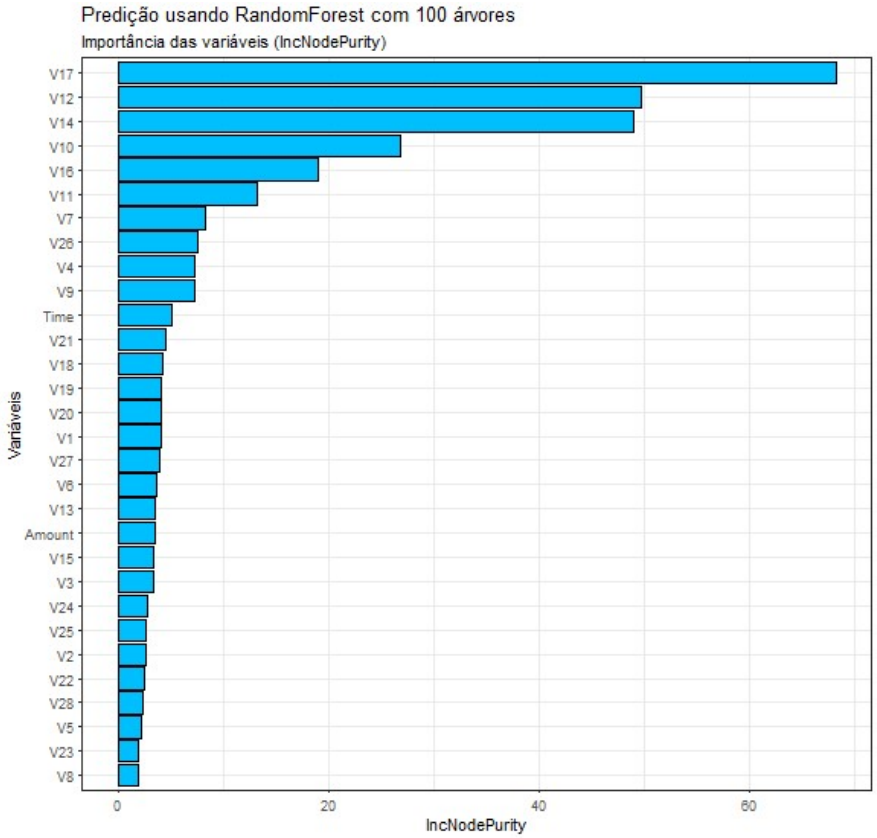

Figura 12. Gráfico da importância dos atributos gerado pelo Random forest

v.4. Neural Net

Os resultados obtidos pela rede neural não levaram em consideração a totalidade do conjunto de treinamento. Aparentemente, a presença minoritária das fraudes não foi capaz de ajustar os pesos dados às conexões entre os neurônios.

Dessa forma, alguns testes foram feitos selecionando subconjuntos muito menores do que o treinamento original que era da ordem de 200 mil observações. Mesmo com um consumo elevado de memória, cerca de 12 Gb, as melhores respostas encontradas levaram em consideração apenas 300 observações, em um processamento de quase uma hora.

Para estas especificações, a depender do critério de arredondamento (*threshold*), foi possível identificar corretamente entre 145 e 149 das 160 fraudes no conjunto treino e cerca de 80% das observações não fraudes foram classificadas corretamente. A próxima subseção irá mostrar que esses resultados são muito inferiores aos obtidos pelos métodos anteriores.

v.5. Seleção do *threshold* baseado no custo

O *threshold* padrão (*Default*) é 0,5, entretanto esse pode não ser o melhor para cada um dos algoritmos de mineração.

v.5.1. CART

Observamos que o custo dos falsos positivos, estimado em €15 por observação, e dos falsos negativos, dado pelo valor da transação, em relação aos *thresholds* forma platôs, Figura 13. O mesmo ocorre quando relacionamos a quantidade total de falsos e o custo total com os thresholds, Figura 14.

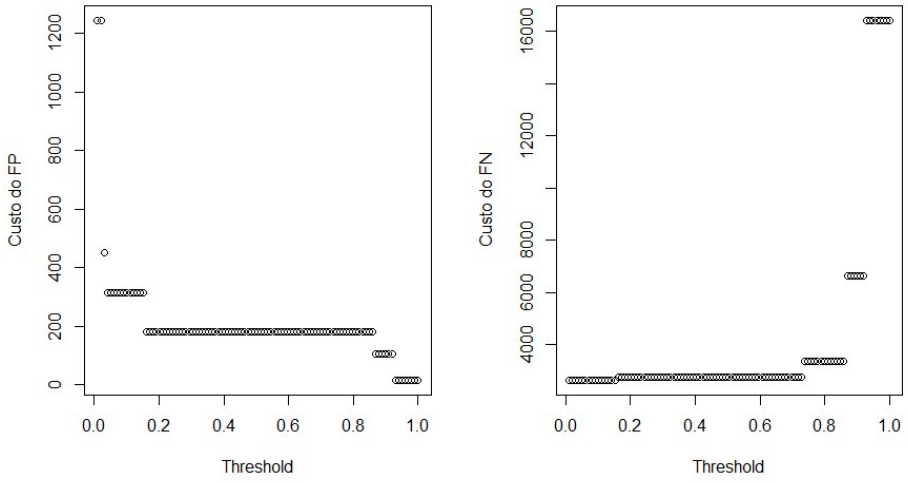

Figura 13. Gráfico de FP e FN x *thresholds (CART)*

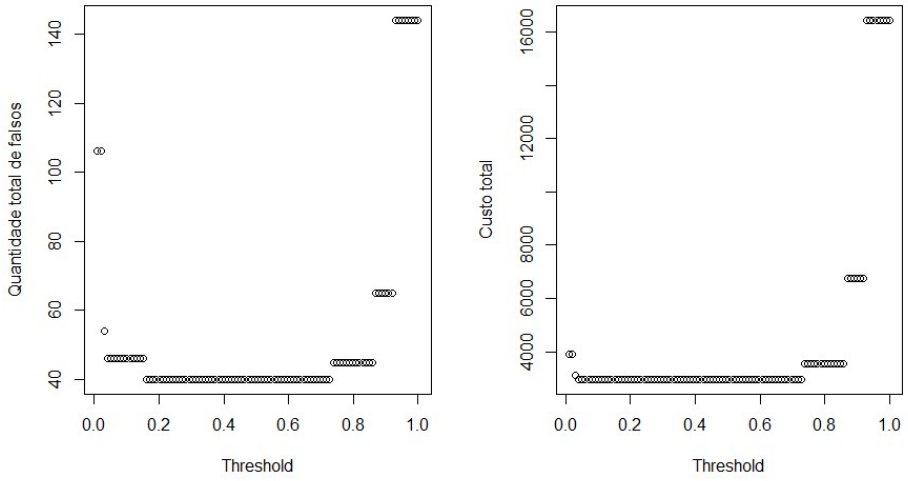

Figura 14. Gráfico da Quantidade total de falsos e Custo total x *thresholds (CART)*

O gráfico da quantidade total de falsos e do custo total, indica um mínimo com um total de falsos igual a 40 e um custo total de € 2.957,96 para um t*hreshold de* 0,16.

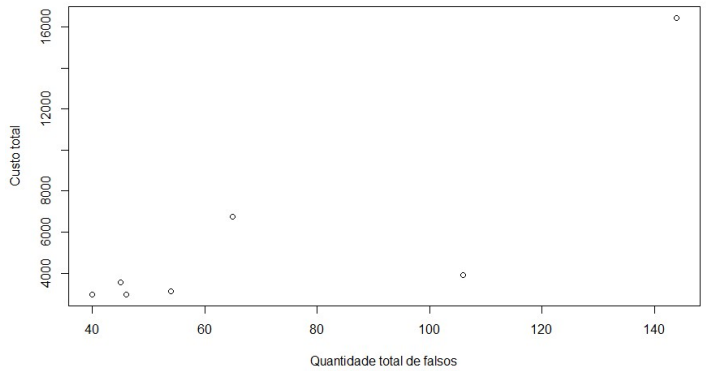

Figura 15. Gráfico da Quantidade total de falsos e Custo total x *thresholds*

É interessante perceber que a matriz confusão para os *thresholds* de 0,5 e 0,16 não se alteram. Constatamos que ambos possuem os mesmos resultados e que seria indiferente escolher entre um e outro, conforme podemos observar na Figura 14. Existe também a opção de escolha de outros *thresholds* que possuem o mesmo custo, mas uma quantidade total de falsos diferentes, como podemos observar na Figura 15.

v.5.2. Random Forest

Novamente construímos os gráficos de custos com relação aos falsos positivos, falsos negativos, um mensurando o total de falsos e outro para o custo total relacionado aos *thresholds*. Percebe-se que, diferente dos platôs observados no CART, aqui temos uma dependência maior do *threshold.*

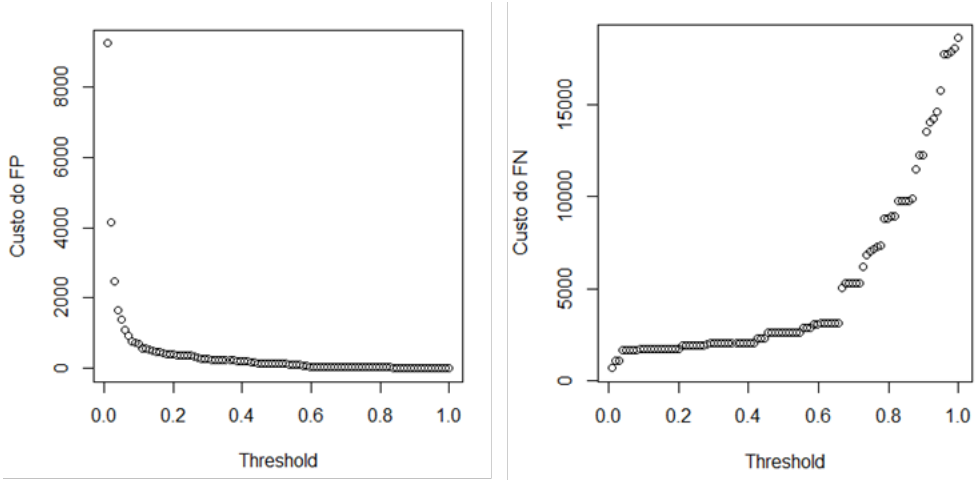

Figura 16. Gráfico de FP e FN x *thresholds (Random Forest)*

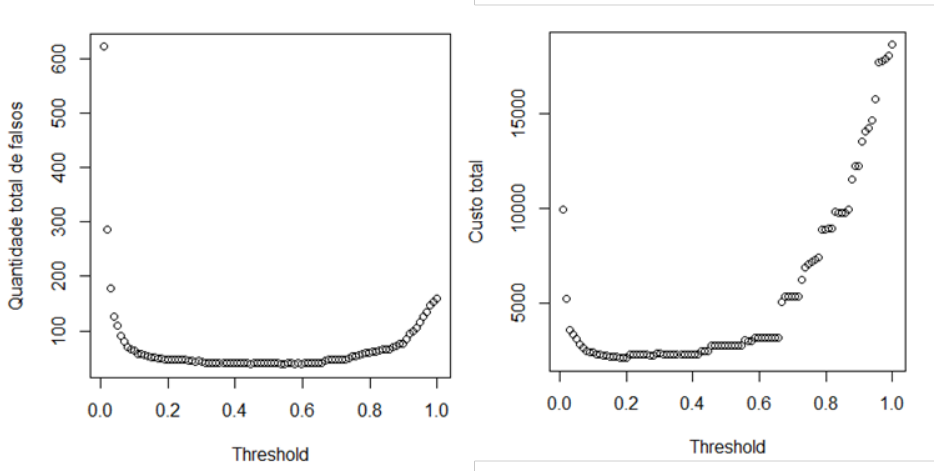

Figura 17. Gráfico da Quantidade total de falsos e Custo total x *thresholds (Random Forest)*

O gráfico da quantidade total de falsos e do custo total, indica um mínimo com um total de falsos igual a 47 e um custo total de € 2.103,10 para um t*hreshold de* 0,2. Existe ainda uma opção com um número menor de falsos, 38, em um t*hreshold* de 0,57, a um custo de € 2.988,50.

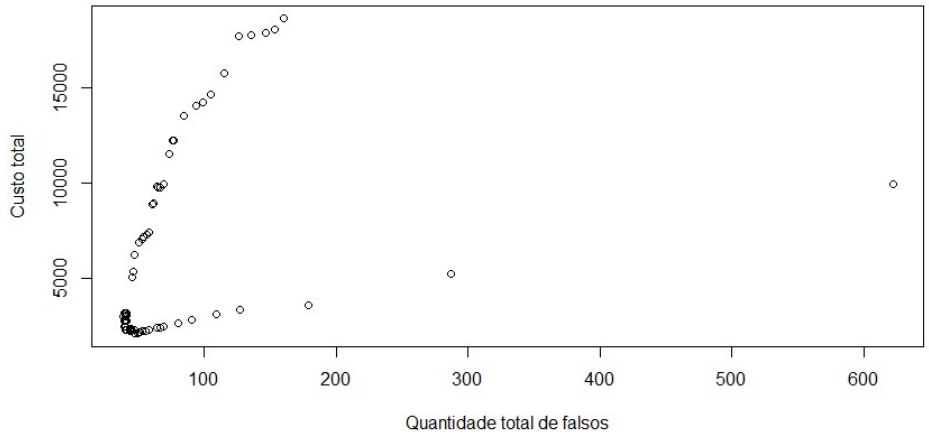

Figura 18. Gráfico da Quantidade total de falsos e Custo total x *thresholds (Random Forest)*

v.5.1. Neural Net

O comportamento dos *thresholds* nas redes neurais é diferente dos métodos citados anteriormente. Como pode ser observado na Figura 19, o aumento desse valor só implica em uma diminuição do número de falsos, sendo que a escolha do valor 1 minimiza os erros. Isso significa que a rede parece já sinalizar com maior precisão o valor 1 para os casos de fraude. Comportamento idêntico em relação ao custo total, fortemente puxado para cima pelo custo dos falsos positivos (€15 por ocorrência).

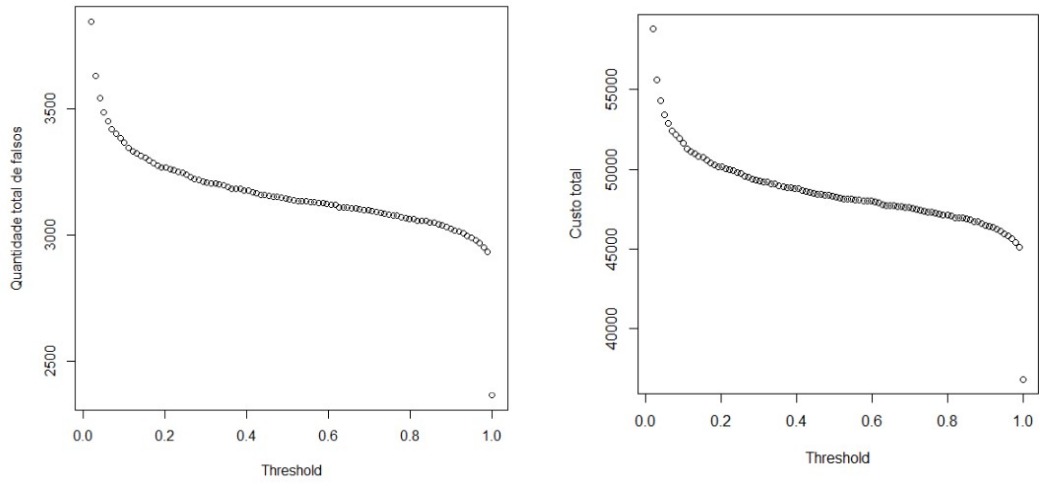

Figura 19. Gráfico da Quantidade total de falsos e Custo total x *thresholds (Neural Net)*

Neste ponto, como pode ser visto na Figura 20, evidenciamos o pior desempenho das redes neurais nesta tarefa. O gráfico da quantidade total de falsos e do custo total, indica um mínimo com um total de falsos igual a 2366 e um custo total de € 36.838,20 para um t*hreshold de* 1, um custo quase 18 vezes maior que o melhor resultado do *Random Forest*.

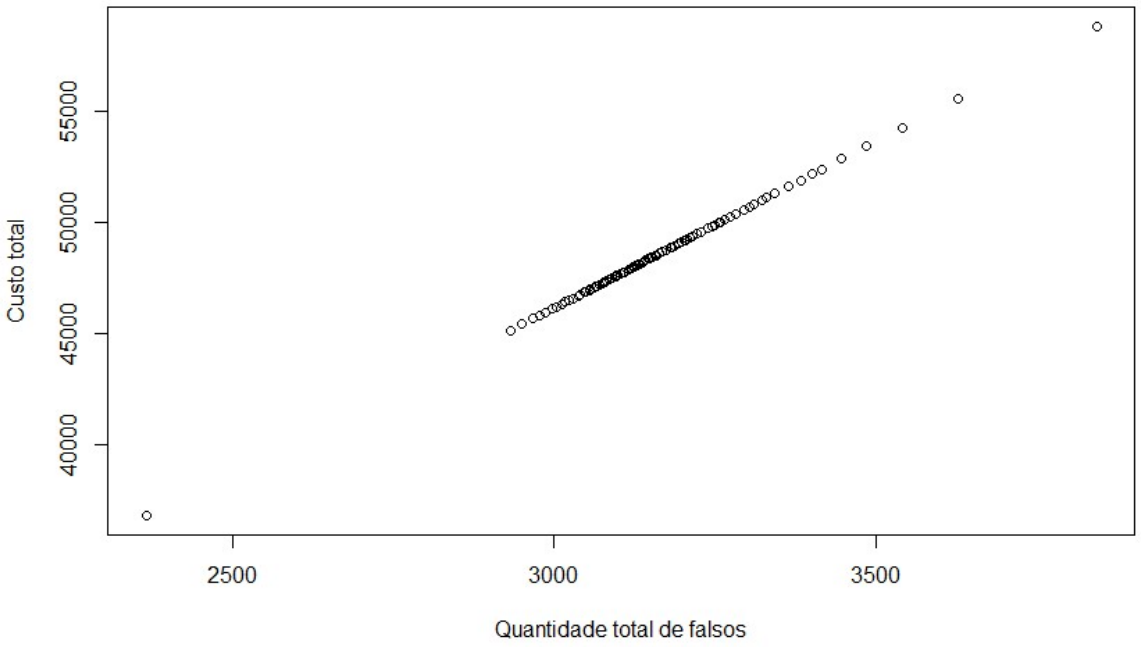

Figura 20. Gráfico da Quantidade total de falsos e Custo total x *thresholds (Neural Net)*

vi. Oportunidades para pesquisas futuras e o que eu gostaria de saber se pudesse fazer novamente o projeto.

Em nossas pesquisas, seja na literatura, em fóruns ou mesmo acompanhando os seminários dos programas de pós-graduação da UNIFESP/ITA, nos deparamos com outros métodos que poderiam trazer outros resultados para comparação, como Naive-Bayes ou Xgboost. Outros métodos com bons resultados poderiam colaborar para uma busca multimetodológica na separação dos dados.

Outro ponto seria buscar adaptações com as redes neurais de modo a encontrar um resultado mais razoável. Nos parece que o conjunto desbalanceado prejudicou seu desempenho, mas acreditamos que essa metodologia poderia gerar ainda mais discussões.

De todo modo, um passo natural seria estudar uma previsibilidade baseada em séries temporais, dando um peso maior às transações recentes sem esquecer completamente os dados históricos.

Acreditamos que buscas nesse sentido poderiam contribuir com passos e pesquisas futuras.

Em linhas gerais, um controle sobre a linguagem R nos parece ser a questão principal entre todas as dificuldades que se mostraram ao longo do processo.

vii. Conclusões

Analisando os resultados encontrados dentro da própria plataforma do Kaggle, o formato final deste trabalho nos parece bastante satisfatório. Primeiramente porque acreditamos que as questões iniciais foram respondidas ou ao menos uma parte delas.

O número de fraudes pode ser mitigado utilizando estratégias baseadas em árvores. Dependendo do tamanho do conjunto de dados e do poder computacional disponível, a escolha deve ficar entre os algoritmos Cart e Random Forest.

Podemos utilizar esses algoritmos conhecidos para essa problemática, mas nos parece claro que, a depender do conjunto de dados, métodos diferentes podem performar melhor ou pior. Para se ter o controle disso é preciso dominar as métricas que avaliam os métodos, como a AUCPR.

As informações que um gestor da área de fraudes bancárias precisaria nos parecem estar concentradas na relação número de fraudes e custo total. O custo da imagem da instituição financeira, desgastado por um possível bloqueio recorrente de transações, nos parece ser algo difícil de dimensionar e, por esse motivo, devem capitanear as decisões sobre qual tipo de método utilizar.

Por fim, cabe também refletir sobre a metodologia na qual esta disciplina é

baseada. A forma como escolhemos o problema, convivemos com suas nuances e chegamos neste ponto nos deixa certos de termos encontrado muitas outras respostas além das que estão apresentadas aqui. Nos parece o caminho correto, problemas, perguntas e respostas.

viii. Bibliografia

AKILA, S.; SRINIVASULU, R.U.; Cost-sensitive Risk indced Bayesian Inference Bagging (RIBIB) for credit card fraud. Journal of Computational Science. 2018. 27, pages 247-254.

ALESKEROV, Emin; FIEISLEBEN, Bernd; RAO, Bharat. CARDWATCH: A Neural Network Based Database Mining System for Credit Card Fraud Detection. Computational Intelligence for Financial Engineering (CIFEr), Proceedings of the IEEE/IAFE. New York City, NY,USA,1997. Disponível em: https://ieeexplore.ieee.org/document/618940/references#references Acesso em: 26/08/2019.

Breiman,L.,Friedman,J.H.,Olshen,R.A.,andStone,C.J. (1984). ClassificationandRegressionTrees.Chapman& Hall/CRC.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). The kdd process for extracting useful knowledge from volumes of data. Communications of the ACM, 39(11):27–34.

GADI, Manoel F. A. Uma comparação de métodos de classificação aplicados à detecção de fraudes de cartão de crédito.2008, 182 f. Dissertação. (Mestre em Ciências) – Instituto de Matemática e Estatística da Universidade de São Paulo,São Paulo, 2008.

GÜNTHER ,Fraukeand; FRITSCH, Stefan. neuralnet: Training of Neural Networks. The R Journal. .2010,2:1,pages 30-38. Disponível em: https://journal.r-project.org/archive/2010/RJ-2010-006/index.html Acesso em: 09/09/2019.

Haykin S., *Neural Networks ~ A Comprehensive Foundation.* Macmillan College Publishing, 1994.

Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome. The Elements of Statistical Learning (2nd ed.). Springer, 2008.

Jakitas, Renato. Fraudes em cartão de credito já passam de 920 mil no pais desde o inicio do ano. O Estado de S.Paulo, São Paulo, 28 de setembro de 2018. Economia & Negócios. Disponível em: < https://economia.estadao.com.br/noticias/geral,fraudes-em-cartao-de-credito-ja-passam-de-920-mil-no-pais-desde-o-inicio-do-ano,70002517637>. Acesso em: 02/12/2019.

Williams G. (2011) Decision Trees. In: Data Mining with Rattle and R. Use R. Springer, New York, NY

TORGO, L. Data Mining with R: learning by case studies. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/3518820/mod_resource/content/1/DataMiningWithR.pdf>. Acesso em 26/11/2019.

ix. Anexos

Código relacionado ao CART

```
# Resolução de problemas
# Danielle Guanaes
# Usando o CARET para fazer o CV

setwd("D:/PO/Resolucao")

library(corrplot)
library(dplyr)
library (caret)

### Leitura dos dados
dados <- read.csv("creditcard.csv",stringsAsFactors=FALSE)
dim(dados)
str(dados)
summary(dados)

### Classe minoritária (fraudes)
sum(dados$Class == 1)
fraude <- sum(dados$Class == 1)/ nrow(dados)
sprintf("%1.2f%%", 100*fraude)

### Missing data: não há dados faltantes
ausentes <- sum(is.na(dados))/(nrow(dados)*ncol(dados))
sprintf("%1.3f%%", 100*ausentes)

### Amostras para treinamento e teste do modelo para dados
set.seed(13)
str(dados)
rowIndex <- createDataPartition(dados$Class, p = 0.7, list = FALSE)
treino <- dados[rowIndex,]
teste  <- dados[-rowIndex,]

str(treino)
str(teste)
table(treino$Class)
table(teste$Class)

#### Árvore de decisão - CART
library(rpart)
library (caret)
library(rattle)
library(rpart.plot)
library(RColorBrewer)

## Create a Full Tree
#caret.control <- trainControl(method='cv', number=10)
#rpart.cv                                                                    <-
train(as.factor(Class)~.,data=treino,method='rpart',trcontrol=caret.control,
control=rpart.control(minsplit=2, cp=0)

#cost <- matrix(c(0,1,1,0),byrow=TRUE, nrow=2)
#tree_full     <-     rpart(as.factor(Class)~.,data=treino,    parms=list(loss=cost),
control=rpart.control(minsplit=2, cp=0))
tree_full <- rpart(as.factor(Class)~.,data=treino, control=rpart.control(minsplit=2,
cp=0))
plotcp(tree_full) # visualize cross-validation results - cp=0.011, erro menor que
0.4
fancyRpartPlot(tree_full, caption = NULL) #there may be some overplotting

## Create Pruned Tree
ptree<- prune(tree_full, cp=0.011) #fica plato
prp(ptree,type = 4, extra = 4,box.palette = "auto",faclen = 0)
fancyRpartPlot(ptree, caption = NULL,main="CART podado")
```

```
## Variable Importance
par (mfrow=c(1,1))
library(vip)
ptreeImp <- ptree$variable.importance
dotplot(ptreeImp,      main="Importância      dos      atributos      -      CART",
ylab="Atributos",xlab="Importância")

## Desempenho do treino
# Recall x Precision Recall
library(ROCR)
pred_treinoptree<-predict(ptree,treino,"prob")
pr_treinoptree<-prediction(pred_treinoptree[,2],treino$Class)
perf_treinoptree<-performance(pr_treinoptree,"prec", "rec")
custo<-performance(pr_treinoptree, "fnr")
plot(perf_treinoptree)

# AUC Recall x Precision
x_treinoptree<- perf_treinoptree@x.values[[1]]
y_treinoptree<- perf_treinoptree@y.values[[1]]
require(Bolstad2)
auc<- sintegral(x_treinoptree,y_treinoptree)$int
auc

## Desempenho do teste
# Recall x Precision Recall
library(ROCR)
pred_testeptree<-predict(ptree,teste,"prob")
pr_testeptree<-prediction(pred_testeptree[,2],teste$Class)
perf_testeptree<-performance(pr_testeptree,"prec", "rec")
plot(perf_testeptree)

# AUC Recall x Precision
x_testeptree<- perf_testeptree@x.values[[1]]
y_testeptree<- perf_testeptree@y.values[[1]]
require(Bolstad2)
auc<- sintegral(x_treinoptree,y_treinoptree)$int
auc

## Escolha dos melhores parâmetros
resultado  <- data.frame(teste$Amount,pred_testeptree[,2],teste$Class)
colnames(resultado)[1] <- "Amount"
colnames(resultado)[2] <- "CART"
colnames(resultado)[3] <- "Class"


Calculo_Custo <- function(resultado) {

  n=100

  fp_CART <- function(resultado, threshold) {
    sum(resultado$CART >= threshold & resultado$Class == 0)
  }

  fp_cost_CART <- function(resultado, threshold) {
    sum(ifelse (resultado$CART >= threshold & resultado$Class == 0, 15,0))
  }

  fn_CART <- function(resultado, threshold) {
    sum(resultado$CART < threshold & resultado$Class == 1)
  }

  fn_cost_CART <- function(resultado, threshold) {
    sum(ifelse  (resultado$CART  <  threshold  &  resultado$Class==1,  resultado$Amount,
0))
  }

  resposta <- data.frame(threshold = seq(0,1,length.out=n))
  resposta$fp_CART  <-  sapply(resposta$threshold,  function  (th)  fp_CART(resultado,
```

```
th))
  resposta$fp_cost_CART      <-      sapply(resposta$threshold,      function      (th)
fp_cost_CART(resultado, th))
  resposta$fn_CART  <-  sapply(resposta$threshold,  function  (th)  fn_CART(resultado,
th))
  resposta$fn_cost_CART      <-      sapply(resposta$threshold,      function      (th)
fn_cost_CART(resultado, th))

  return(resposta)
}

Custo_total <- Calculo_Custo(resultado)

Custo_total$FT <- Custo_total$fn_CART + Custo_total$fp_CART
Custo_total$CT <- Custo_total$fp_cost_CART + Custo_total$fn_cost_CART

summary(Custo_total)

plot(Custo_total$threshold, Custo_total$fp_cost_CART, xlab="Threshold", ylab="Custo
do FP")
#thresold =0 faz com o gráfico estoure
idx_threshold <- which.min(Custo_total$CT)
Custo_total0 <- Custo_total[-1, ]

par (mfrow=c(1,2))
plot(Custo_total0$threshold,          Custo_total0$fp_CART,          xlab="Threshold",
ylab="Quantidade de Falsos Positivos")
plot(Custo_total0$threshold,          Custo_total0$fn_CART,          xlab="Threshold",
ylab="Quantidade de Falsos Negativos")

plot(Custo_total0$threshold,       Custo_total0$fp_cost_CART,       xlab="Threshold",
ylab="Custo do FP")
#lines(Custo_total0[,1], Custo_total0[,3],col="red")
plot(Custo_total0$threshold,       Custo_total0$fn_cost_CART,       xlab="Threshold",
ylab="Custo do FN")

par (mfrow=c(1,2))
plot(Custo_total0$threshold,  Custo_total0$FT,  xlab="Threshold",  ylab="Quantidade
total de falsos")
plot(Custo_total0$threshold, Custo_total0$CT,xlab="Threshold", ylab="Custo total")

par (mfrow=c(1,1))
plot(Custo_total0$FT, Custo_total0$CT,xlab="Quantidade total de falsos", ylab="Custo
total")

FT_min<-Custo_total0[which.min(Custo_total0$FT),6]
CT_min<-Custo_total0[which.min(Custo_total0$FT),7]
th_min<-Custo_total0[which.min(Custo_total0$FT),1]

## Calculo das acurácias com thresholds diferentes
library(SDMTools)
th50 <- accuracy(teste$Class, pred_testeptree[,2], threshold = 0.5)
th50
cm50 <- confusion.matrix(teste$Class, pred_testeptree[,2], threshold = 0.5)
cm50

th16 <- accuracy(teste$Class, pred_testeptree[,2], threshold = 0.1616)
th16
cm16 <- confusion.matrix(teste$Class, pred_testeptree[,2], threshold = 0.1616)
cm16
```

Código relacionado ao RANDOM FOREST

```
# Resolução de problemas
# Thiago Siqueira

setwd("C:/R_arquivos")

library(corrplot)
library(dplyr)
library (caret)
library(randomForest)

### Leitura dos dados
dados <- read.csv("creditcard.csv",stringsAsFactors=FALSE)
dim(dados)
str(dados)
summary(dados)

### Missing data: não há dados faltantes
ausentes <- sum(is.na(dados))/(nrow(dados)*ncol(dados))
sprintf("%1.3f%%", 100*ausentes)

### Amostras para treinamento e teste do modelo para dados2
set.seed(13)
str(dados)
rowIndex <- createDataPartition(dados$Class, p = 0.7, list = FALSE)
treino <- dados[rowIndex,]
teste  <- dados[-rowIndex,]

str(treino)
str(teste)
table(treino$Class)
table(teste$Class)


### Leitura dos dados
dados <- read.csv("creditcard.csv",stringsAsFactors=FALSE)
dim(dados)
str(dados)
summary(dados)

### Missing data: não há dados faltantes
ausentes <- sum(is.na(dados))/(nrow(dados)*ncol(dados))
sprintf("%1.3f%%", 100*ausentes)

### Amostras para treinamento e teste do modelo para dados2
set.seed(13)
str(dados)
rowIndex <- createDataPartition(dados$Class, p = 0.7, list = FALSE)
treino <- dados[rowIndex,]
teste  <- dados[-rowIndex,]

str(treino)
str(teste)
table(treino$Class)
table(teste$Class)


## Primeiro exemplo RandonForest
##  Adaptado  de  https://www.kaggle.com/gpreda/credit-card-fraud-detection-with-rf-
auc-0-93



n <- names(treino)
rf.form <- as.formula(paste("Class ~", paste(n[!n %in% "Class"], collapse = " + ")))

treino.rf <- randomForest(rf.form,treino,ntree=100,importance=T)
```

```
## Importancia das variáveis


varimp <- data.frame(treino.rf$importance)

vi1      <-      ggplot(varimp,      aes(x=reorder(rownames(varimp),IncNodePurity),
y=IncNodePurity)) +
  geom_bar(stat="identity", fill="deepskyblue", colour="black") +
  coord_flip() + theme_bw(base_size = 8) +
  labs(title="Predição usando RandomForest com 100 árvores", subtitle="Importância
das variáveis (IncNodePurity)", x="Variáveis", y="IncNodePurity")

vi2 <- ggplot(varimp, aes(x=reorder(rownames(varimp),X.IncMSE), y=X.IncMSE)) +
  geom_bar(stat="identity", fill="darkmagenta", colour="black") +
  coord_flip() + theme_bw(base_size = 8) +
  labs(title="Predição usando RandomForest com 100 árvores", subtitle="Importância
das variáveis (%IncMSE)", x="Variáveis", y="%IncMSE")

grid.arrange(vi1, vi2, ncol=2)

##Matriz de confusão

teste$previsto <- predict(treino.rf ,teste)

tst <- data.frame(round(teste$previsto,0), teste$Class)
opts <-  c("Predicted", "True")
names(tst) <- opts
cf <- plyr::count(tst)
cf[opts][cf[opts]==0] <- "Not Fraud"
cf[opts][cf[opts]==1] <- "Fraud"

ggplot(data =  cf, mapping = aes(x = True, y = Predicted)) +
  labs(title = "Matriz de confusão") +
  geom_tile(aes(fill = freq), colour = "grey") +
  geom_text(aes(label = sprintf("%1.0f", freq)), vjust = 1) +
  scale_fill_gradient(low = "lightblue", high = "blue") +
  theme_bw() + theme(legend.position = "none")

##Função para Calcular ROC E AUC
calculate_roc <- function(teste, cost_of_fp, cost_of_fn, n=100) {

  tp <- function(teste, threshold) {
    sum(teste$previsto2000_mtry10 >= threshold & teste$Class == 1)
  }

  fp <- function(teste, threshold) {
    sum(teste$previsto2000_mtry10 >= threshold & teste$Class == 0)
  }

  tn <- function(teste, threshold) {
    sum(teste$previsto2000_mtry10 < threshold & teste$Class == 0)
  }

  fn <- function(teste, threshold) {
    sum(teste$previsto2000_mtry10 < threshold & teste$Class == 1)
  }

  tpr <- function(teste, threshold) {
    sum(teste$previsto2000_mtry10 >= threshold & teste$Class == 1) / sum(teste$Class
== 1)
  }

  fpr <- function(teste, threshold) {
    sum(teste$previsto2000_mtry10 >= threshold & teste$Class == 0) / sum(teste$Class
== 0)
  }
```

```
  cost <- function(teste, threshold, cost_of_fp, cost_of_fn) {
  #  sum(teste$previsto2000_mtry10 >= threshold & teste$Class == 0) * 15 +
      sum(teste$previsto2000_mtry10 < threshold & teste$Class == 1) * 1
  }
  fpr <- function(teste, threshold) {
    sum(teste$previsto2000_mtry10 >= threshold & teste$Class == 0) / sum(teste$Class
== 0)
  }

  threshold_round <- function(value, threshold)
  {
    return (as.integer(!(value < threshold)))
  }
  #calculate                                                                    AUC
(https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curv
e)
  auc_<- function(teste, threshold) {
    auc(teste$Class, threshold_round(teste$previsto_2000mtry10,threshold))
  }

  roc_2000mtry10 <- data.frame(threshold = seq(0,1,length.out=n), tpr=NA, fpr=NA)
  roc_2000mtry10$tp <- sapply(roc_2000mtry10$threshold, function(th) tp(teste, th))
  roc_2000mtry10$fp <- sapply(roc_2000mtry10$threshold, function(th) fp(teste, th))
  roc_2000mtry10$tn <- sapply(roc_2000mtry10$threshold, function(th) tn(teste, th))
  roc_2000mtry10$fn <- sapply(roc_2000mtry10$threshold, function(th) fn(teste, th))
  roc_2000mtry10$tpr  <-  sapply(roc_2000mtry10$threshold,  function(th)  tpr(teste,
th))
  roc_2000mtry10$fpr  <-  sapply(roc_2000mtry10$threshold,  function(th)  fpr(teste,
th))
  roc_2000mtry10$cost  <-  sapply(roc_2000mtry10$threshold,  function(th)  cost(teste,
th, cost_of_fp, cost_of_fn))
  #roc_2000mtry10$auc <-   sapply(roc_2000mtry10$threshold, function(th) auc_(teste,
th))

  return(roc_2000mtry10)
}
## Com a ROC feita, construir a PR

PR_teste<-roc_2000mtry10
names(PR_teste)  <-  c("threshold",  "precision",  "recall",  "tp",  "fp",  "tn",  "fn",
"cost", "auc" )
PR_teste$precision<-PR_teste$tp/(PR_teste$tp + PR_teste$fp)
PR_teste$recall<-PR_teste$tp/(PR_teste$tp + PR_teste$fn)
PR_teste$cost_fp<-(PR_teste$fp*15)

##Função para Plotar ROC e AUC

plot_roc <- function(roc, threshold, cost_of_fp, cost_of_fn) {
  library(gridExtra)

  norm_vec <- function(v) (v - min(v))/diff(range(v))

  idx_threshold = which.min(abs(roc$threshold-threshold))

  col_ramp <- colorRampPalette(c("green","orange","red","black"))(100)
  col_by_cost <- col_ramp[ceiling(norm_vec(roc$cost)*99)+1]
  p_roc <- ggplot(roc, aes(fpr,tpr)) +
    geom_line(color=rgb(0,0,1,alpha=0.3)) +
    geom_point(color=col_by_cost, size=2, alpha=0.5) +
    labs(title = sprintf("ROC")) + xlab("FPR") + ylab("TPR") +
    geom_hline(yintercept=roc[idx_threshold,"tpr"], alpha=0.5, linetype="dashed") +
    geom_vline(xintercept=roc[idx_threshold,"fpr"], alpha=0.5, linetype="dashed")

  p_auc <- ggplot(roc, aes(threshold, auc)) +
    geom_line(color=rgb(0,0,1,alpha=0.3)) +
    geom_point(color=col_by_cost, size=2, alpha=0.5) +
    labs(title = sprintf("AUC")) +
    geom_vline(xintercept=threshold, alpha=0.5, linetype="dashed")
```

```
  p_cost <- ggplot(roc, aes(threshold, cost)) +
    geom_line(color=rgb(0,0,1,alpha=0.3)) +
    geom_point(color=col_by_cost, size=2, alpha=0.5) +
    labs(title = sprintf("cost function")) +
    geom_vline(xintercept=threshold, alpha=0.5, linetype="dashed")

  sub_title <- sprintf("threshold at %.2f - cost of FP = %d, cost of FN = %d",
threshold, cost_of_fp, cost_of_fn)
  #
  grid.arrange(p_roc, p_auc, p_cost, ncol=2,sub=textGrob(sub_title, gp=gpar(cex=1),
just="bottom"))
}

## Rotina para calcular AUC

teste$previsto <- predict(treino.rf ,teste)

roc_nn <- calculate_roc(teste, 15, 15, n = 100)

mincost <- min(roc$cost)
roc %>%
  mutate(
    auc = ifelse(cost == mincost,
                 cell_spec(sprintf("%.5f", auc), "html", color = "green", background
= "lightblue", bold = T),
                 cell_spec(sprintf("%.5f", auc), "html", color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center")
%>%
  scroll_box(height = "600px")



## PLotando a curva PR (comparando o braco com o ROCR)
pred <- prediction(teste$previsto200,teste$Class)
perf <- performance(pred,"prec","rec")
plot(perf, xlim=c(0,1), ylim=c(0,1))

par(new=T)

plot(PR_teste$recall,PR_teste$precision,
     main="Curva Precision Recall",
     ylab=" ", xlab=" ", xlim=c(0,1), ylim=c(0,1),
     col="blue")
par(new=F)


AUC_pacote <- sintegral(perf@x.values[[1]],perf@y.values[[1]], 4096)$int


## Segundo exemplo RandonForest (ntree=200)


n <- names(treino)
rf.form <- as.formula(paste("Class ~", paste(n[!n %in% "Class"], collapse = " + ")))

treino200.rf <- randomForest(rf.form,treino,ntree=200,importance=T)




## Terceiro exemplo RandonForest (ntree=200)

Sys.time()
n <- names(treino)
rf.form <- as.formula(paste("Class ~", paste(n[!n %in% "Class"], collapse = " + ")))
```

```
treino2000_mtry10.rf <- randomForest(rf.form,treino,ntree=2000,importance=T,mtry=10)
Sys.time()


teste$previsto2000_mtry10 <- predict(treino2000_mtry10.rf ,teste)


library(Bolstad2)
pred <- prediction(testset$previsto2000_mtry10,teste$Class)
perf <- performance(pred,"prec","rec")
plot(perf, xlim=c(0,1), ylim=c(0,1))
AUC_pacote2000_mtry10 <- sintegral(perf@x.values[[1]],perf@y.values[[1]], 4096)$int
```

Código relacionado a Rede Neural

```
library(neuralnet)
library(caret)
library(nnet)
library(tibble)

attach(creditcard)


#Min-Max NORMALIZAcao DOS DADOS


normalize<-function(x){
  return ((x-min(x))/(max(x)-min(x)))
}

maxmindf<-as.data.frame(lapply(creditcard[,1:30],normalize))
creditcard<-cbind(maxmindf,Class)


#SEPARAR 30 E 70 OBSERVACOES 1

set.seed(13)
str(creditcard)
rowIndex <- createDataPartition(creditcard$Class, p = 0.7, list = FALSE)

#DADOS DE TREINO E TESTE

#CONJUNTO TREINO
trainset1 <- creditcard[rowIndex,] #recebe 30 da particÃ£o aleatÃ³ria anterioir
trainsetUM<-trainset1[trainset1$Class>0,]# aloca todas as observaÃ§Ãµes 1
trainsetZEROS<-trainset1[trainset1$Class==0,]# aloca todas as observaÃ§Ãµes 0

trainsetUM.aleatorios <- trainsetUM[sample(1:nrow(trainsetUM),200),]# seleciona 150
observaÃ§Ãµes 1 aleatÃ³riamente
trainsetZEROS.aleatorios <- trainsetZEROS[sample(1:nrow(trainsetZEROS),400),]#
seleciona 150 observaÃ§Ãµes 0 aleatÃ³riamente


trainset<-rbind(trainsetUM.aleatorios[,],trainsetZEROS.aleatorios[,])#forma um
conjunto de treino com iguais observaÃ§Ãµes de zeros e uns


#CONJUNTO TESTE

testset <- creditcard[-rowIndex,]#conjunto de teste da partiÃ§Ã£o 70/30


#TREINAMENTO REDE NEURAL

Sys.time()
nn1<-neuralnet(Class~.,data=trainset,hidden= c(3,2,1),
linear.output=FALSE,threshold=0.0085,rep=100,err.fct="ce")


#TESTE/PREVISÃ O DA REDE NEURAL

testsetresult<-predict(nn1,testset) #Computa o conjunto de teste com a rede neural e
faz a previsÃ£o

predict.testset<-cbind(testset[,31],testsetresult)#compara o previsto com o
observado


#Acuracia// matriz de confusao
results<-data.frame(actual = testset$Class, prediction = testsetresult) #semelhante
ao "predict.testset" porÃ©m Ã© um data.frame com colunas nomeadas
roundedresults<-sapply(results,round,digits=0)# arrendonda os valores para 0 ou 1
```

```
roundedresultsdf<-data.frame(roundedresults) # cria um data frame com as valores
arredondados
attach(roundedresultsdf)
table(actual,prediction)#gera matriz de confusÃ£o
Sys.time()

A=as.data.frame(testsetresult)
testset$previstonn <-A$V1

#gráficos

resultado  <- data.frame(testset$Amount,testset$previstonn,testset$Class)
colnames(resultado)[1] <- "Amount"
colnames(resultado)[2] <- "Rede_Neural"
colnames(resultado)[3] <- "Class"

Calculo_Custo <- function(resultado) {

  n=100

  fp_Rede_Neural <- function(resultado, threshold) {
    sum(resultado$Rede_Neural >= threshold & resultado$Class == 0)
  }

  fp_cost_Rede_Neural <- function(resultado, threshold) {
    sum(ifelse (resultado$Rede_Neural >= threshold & resultado$Class == 0, 15,0))
  }

  fn_Rede_Neural <- function(resultado, threshold) {
    sum(resultado$Rede_Neural < threshold & resultado$Class == 1)
  }

  fn_cost_Rede_Neural <- function(resultado, threshold) {
    sum(ifelse (resultado$Rede_Neural < threshold & resultado$Class==1,
resultado$Amount, 0))
  }

  resposta <- data.frame(threshold = seq(0,1,length.out=n))
  resposta$fp_Rede_Neural <- sapply(resposta$threshold, function (th)
fp_Rede_Neural(resultado, th))
  resposta$fp_cost_Rede_Neural <- sapply(resposta$threshold, function (th)
fp_cost_Rede_Neural(resultado, th))
  resposta$fn_Rede_Neural <- sapply(resposta$threshold, function (th)
fn_Rede_Neural(resultado, th))
  resposta$fn_cost_Rede_Neural <- sapply(resposta$threshold, function (th)
fn_cost_Rede_Neural(resultado, th))

  return(resposta)
}
Custo_total <- Calculo_Custo(resultado)
Custo_total$FT <- Custo_total$fn_Rede_Neural + Custo_total$fp_Rede_Neural
Custo_total$CT <- Custo_total$fp_cost_Rede_Neural + Custo_total$fn_cost_Rede_Neural
summary(Custo_total)
plot(Custo_total$threshold, Custo_total$fp_cost_Rede_Neural, xlab="Threshold",
ylab="Custo do FP")

#thresold =0 faz com o gráfico estoure

idx_threshold <- which.min(Custo_total$CT)
Custo_total1 <- Custo_total[-1 , ]
Custo_total0 <- Custo_total1[-1 , ]

plot.new()
plot(Custo_total0$threshold, Custo_total0$fp_Rede_Neural, xlab="Threshold",
ylab="Quantidade de Falsos Positivos")
plot(Custo_total0$threshold, Custo_total0$fp_cost_Rede_Neural, xlab="Threshold",
ylab="Custo do FP")
#lines(Custo_total[,1], Custo_total[,3],col="red")
plot(Custo_total0$threshold, Custo_total0$fn_Rede_Neural, xlab="Threshold",
```

```
ylab="Quantidade de Falsos Negativos")
plot(Custo_total0$threshold, Custo_total0$fn_cost_Rede_Neural, xlab="Threshold",
ylab="Custo do FN")

plot(Custo_total0$threshold, Custo_total0$FT, xlab="Threshold", ylab="Quantidade
total de falsos")
plot(Custo_total0$threshold, Custo_total0$CT,xlab="Threshold", ylab="Custo total")
plot(Custo_total0$FT, Custo_total0$CT,xlab="Quantidade total de falsos", ylab="Custo
total")




calculate_nn <- function(teste, cost_of_fp, cost_of_fn, n=100) {

  tp <- function(teste, threshold) {
    sum(testset$previstonn >= threshold & testset$Class == 1)
  }

  fp <- function(testset, threshold) {
    sum(testset$previstonn >= threshold & testset$Class == 0)
  }

  tn <- function(testset, threshold) {
    sum(testset$previstonn < threshold & testset$Class == 0)
  }

  fn <- function(testset, threshold) {
    sum(testset$previstonn < threshold & testset$Class == 1)
  }

  tpr <- function(testset, threshold) {
    sum(testset$previstonn >= threshold & testset$Class == 1) / sum(testset$Class ==
1)
  }

  fpr <- function(testset, threshold) {
    sum(testset$previstonn >= threshold & testset$Class == 0) / sum(testset$Class ==
0)
  }

  cost <- function(testset, threshold, cost_of_fp, cost_of_fn) {
    #  sum(testset$previstonn >= threshold & testset$Class == 0) * 15 +
    sum(testset$previstonn < threshold & testset$Class == 1) * 1
  }
  fpr <- function(testset, threshold) {
    sum(testset$previstonn >= threshold & testset$Class == 0) / sum(testset$Class ==
0)
  }

  threshold_round <- function(value, threshold)
  {
    return (as.integer(!(value < threshold)))
  }
  #calculate AUC
(https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curv
e)
  auc_<- function(testset, threshold) {
    auc(testset$Class, threshold_round(testset$previstonn,threshold))
  }

  roc_nn <- data.frame(threshold = seq(0,1,length.out=n), tpr=NA, fpr=NA)
  roc_nn$tp <- sapply(roc_nn$threshold, function(th) tp(testset, th))
  roc_nn$fp <- sapply(roc_nn$threshold, function(th) fp(testset, th))
  roc_nn$tn <- sapply(roc_nn$threshold, function(th) tn(testset, th))
  roc_nn$fn <- sapply(roc_nn$threshold, function(th) fn(testset, th))
  roc_nn$tpr <- sapply(roc_nn$threshold, function(th) tpr(testset, th))
```

```
  roc_nn$fpr <- sapply(roc_nn$threshold, function(th) fpr(testset, th))
  roc_nn$cost <- sapply(roc_nn$threshold, function(th) cost(testset, th, cost_of_fp,
cost_of_fn))
  #roc_nn$auc <-  sapply(roc_nn$threshold, function(th) auc_(testset, th))

  return(roc_nn)
}

roc_nn <- calculate_nn(testset, 15, 15, n = 100)

PR_teste<-roc_nn
names(PR_teste) <- c("threshold", "precision", "recall", "tp", "fp", "tn", "fn",
"auc")
PR_teste$precision<-PR_teste$tp/(PR_teste$tp + PR_teste$fp)
PR_teste$recall<-PR_teste$tp/(PR_teste$tp + PR_teste$fn)
PR_teste$cost_fp<-(PR_teste$fp*15)
```