INTRODUÇÃO A APRENDIZADO DE MÁQUINAS

RELATÓRIO

Gabriel Brito Bastos

gabriel.bastos@poli.ufrj.br

DRE:115036300

Março

2021

Sumário

1.Introdução

No trabalho que se segue, irei relatar as etapas de desenvolvimento de um programa da disciplina de Introdução a Aprendizado de Máquinas, código EEL891, do Departamento da Engenharia Eletrônica e da Computação da UFRJ. Este trabalho consiste em desenvolver um classificador em programa na linguagem Python, para identificar possíveis clientes inadimplentes, ou seja, servir de apoio à decisão de solicitação de um determinado produto bancário, como cartão de crédito, empréstimo pessoal, dentre outros.

Para isto, utilizarei uma base de dados real anonimizada de 20 mil dados cadastrais de clientes que foram rotulados como inadimplentes ou não-inadimplentes para o treinamento do classificador, e também utilizarei uma base de dados de teste para medir a performance do classificador utilizado.

Tal projeto permite trabalhar as etapas de tratamento de dados e análise de ferramentas e funções utilizadas a fim de obter a melhor performance, neste caso, não ponderando o tempo de execução, mas sim a taxa de acerto da base de dados-resposta que deve ser submetida na plataforma *Kaggle*.

2.Código-fonte

A fim de iniciar o projeto, algumas bibliotecas tiveram que ser importadas para permitir e agilizar os procedimentos que devem ser feitos, as bibliotecas seguem no código anexo:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression,LinearRegression
from sklearn.neighbors import KNeighborsRegressor
import matplotlib.pyplot as plt
```

Código 1 - Importação das bibliotecas

A biblioteca *numpy* foi utilizada para tratamento de dados, onde foi utilizado o método *where* que mapeia, para uma condição um dado a ser substituído.

numpy.where(condition[, x, y])

Return elements chosen from x or y depending on condition.

A biblioteca *pandas* foi utilizada para o processamento, importação e exportação dos dados, onde foram utilizados diversos métodos que serão detalhados no decorrer do tratamento dos dados.

Para a biblioteca *sklearn*, foi utilizada os grupos de funções de *preprocessing* para pré-tratamento dos dados, *linear_model* para instanciar os classificadores e *neighbors* para instanciar o classificador KNN.

Para a biblioteca *matplotlib*, foi utilizado o grupo de função do *pyplot*, onde neste é possível utilizar diversas formas de plotagem de gráficos.

2.1 Tratamento de dados

O tratamento dos dados é uma etapa fundamental para o projeto, nesta etapa, interpretamos os dados de entrada e verificamos o que é necessário fazer para que este dado se adeque a forma de entrada do classificador.

O primeiro passo a ser feito é importar os dados, para isso, utilizei a função read_csv da biblioteca *pandas*:

```
#--------------------IMPORTACAO DOS DADOS DE ENTRADA ----------------------------

dadosTreinamento = pd.read_csv("conjunto_de_treinamento.csv", index_col = 0)
dadosTeste = pd.read_csv("conjunto_de_teste.csv", index_col = 0)
dadosExemplo = pd.read_csv("exemplo_arquivo_respostas.csv", index_col = 0)
print("Arquivos importados com sucesso\n")
```

Código 2 - Importação da tabela de dados

Em seguida, deve ser iniciado o tratamento dos dados, inicialmente utilizei a função *fillna* da biblioteca *pandas* para preencher com o numeral zero, os campos 'NaN' (*Not a number*, tradução: não é um número):

```
#--------------------PREENCHIMENTO DOS DADOS NULOS/NAN0--------------------------

dadosTreinamento = dadosTreinamento.fillna(0)
dadosTeste = dadosTeste.fillna(0)
```

Código 3 - Preenchimento dos dados NaN com '0'

Então, sigo para a análise das colunas/variáveis da tabela de dados e utilizando o método *drop*, retiro as colunas que não contém dados e colunas que não são pertinentes à análise:

```
atributos_para_retirar = ['grau_instrucao', 'estado_onde_nasceu', 'estado_onde_reside','possui_telefone_celular', 'qtde_contas_bancarias_especiais', 'estado_onde_trabalha', 'codigo_area_telefone_trabalho', 'codigo_area_telefone_residencial', 'local_onde_reside', 'local_onde_trabalha']


#--------------------RETIRANDO ATRIBUTOS NULOS DOS DADOS------------------------
dadosTreinamento = dadosTreinamento.drop(atributos_para_retirar, axis=1)
dadosTeste = dadosTeste.drop(atributos_para_retirar, axis=1)
```

Código 4 - Retirando as colunas que não são pertinentes à classificação

Após a retirada dos atributos que não são pertinentes à análise, visualizei que algumas colunas possuem um formato de dados a qual não é possível ser analisado, que são dados da forma de palavra, ou letra, comumente conhecido na programação como *string*, então, realizei um procedimento utilizando a função *where* da biblioteca *numpy*, para substituir tais dados por uma representação numérica.

Tal mapeamento ocupou bastante linhas do código e poderia ser substituído por uma função de *encoding.*

```
#-------------------MAPEAMENTO DE RESPOSTAS PARA VALORES NUMERICOS ----------------
dadosTreinamento['possui_telefone_residencial'] = np.where(dadosTreinamento['possui_telefone_residencial'] == 'Y' , 1, dadosTreinamento['possui_telefone_residencial'])
dadosTreinamento['possui_telefone_residencial'] = np.where(dadosTreinamento['possui_telefone_residencial'] == 'N' , 0, dadosTreinamento['possui_telefone_residencial'])
dadosTeste['possui_telefone_residencial'] = np.where(dadosTeste['possui_telefone_residencial'] == 'Y' , 1, dadosTeste['possui_telefone_residencial'])
dadosTeste['possui_telefone_residencial'] = np.where(dadosTeste['possui_telefone_residencial'] == 'N' , 0, dadosTeste['possui_telefone_residencial'])

dadosTreinamento['vinculo_formal_com_empresa'] = np.where(dadosTreinamento['vinculo_formal_com_empresa'] == 'Y' , 1, dadosTreinamento['vinculo_formal_com_empresa'])
dadosTreinamento['vinculo_formal_com_empresa'] = np.where(dadosTreinamento['vinculo_formal_com_empresa'] == 'N' , 0, dadosTreinamento['vinculo_formal_com_empresa'])
dadosTeste['vinculo_formal_com_empresa'] = np.where(dadosTeste['vinculo_formal_com_empresa'] == 'Y' , 1, dadosTeste['vinculo_formal_com_empresa'])
dadosTeste['vinculo_formal_com_empresa'] = np.where(dadosTeste['vinculo_formal_com_empresa'] == 'N' , 0, dadosTeste['vinculo_formal_com_empresa'])

dadosTreinamento['possui_telefone_trabalho'] = np.where(dadosTreinamento['possui_telefone_trabalho'] == 'Y' , 1, dadosTreinamento['possui_telefone_trabalho'])
dadosTreinamento['possui_telefone_trabalho'] = np.where(dadosTreinamento['possui_telefone_trabalho'] == 'N' , 0, dadosTreinamento['possui_telefone_trabalho'])
dadosTeste['possui_telefone_trabalho'] = np.where(dadosTeste['possui_telefone_trabalho'] == 'Y' , 1, dadosTeste['possui_telefone_trabalho'])
dadosTeste['possui_telefone_trabalho'] = np.where(dadosTeste['possui_telefone_trabalho'] == 'N' , 0, dadosTeste['possui_telefone_trabalho'])

dadosTreinamento['sexo'] = np.where(dadosTreinamento['sexo'] == 'M' , 1, dadosTreinamento['sexo'])
dadosTreinamento['sexo'] = np.where(dadosTreinamento['sexo'] == 'F' , 2, dadosTreinamento['sexo'])
dadosTreinamento['sexo'] = np.where(dadosTreinamento['sexo'] == ' ' , 3, dadosTreinamento['sexo'])
dadosTreinamento['sexo'] = np.where(dadosTreinamento['sexo'] == 'N' , -1, dadosTreinamento['sexo'])
```

Código 5 - Mapeamento dos atributos formato string - parte 1

```
dadosTeste['sexo'] = np.where(dadosTeste['sexo'] == 'M' , 1, dadosTeste['sexo'])
```

```
dadosTeste['sexo'] = np.where(dadosTeste['sexo'] == 'F' , 2, dadosTeste['sexo'])
dadosTeste['sexo'] = np.where(dadosTeste['sexo'] == ' ' , 3, dadosTeste['sexo'])
dadosTeste['sexo'] = np.where(dadosTeste['sexo'] == 'N' , -1, dadosTeste['sexo'])
```

```
dadosTreinamento['forma_envio_solicitacao'] = np.where(dadosTreinamento['forma_envio_solicitacao'] == 'inte
rnet' , 0, dadosTreinamento['forma_envio_solicitacao'])
dadosTreinamento['forma_envio_solicitacao'] = np.where(dadosTreinamento['forma_envio_solicitacao'] == 'pres
encial' , 1, dadosTreinamento['forma_envio_solicitacao'])
dadosTreinamento['forma_envio_solicitacao'] = np.where(dadosTreinamento['forma_envio_solicitacao'] == 'corr
eio' , 2, dadosTreinamento['forma_envio_solicitacao'])
dadosTeste['forma_envio_solicitacao'] = np.where(dadosTeste['forma_envio_solicitacao'] == 'internet' , 0, d
adosTeste['forma_envio_solicitacao'])
dadosTeste['forma_envio_solicitacao'] = np.where(dadosTeste['forma_envio_solicitacao'] == 'presencial' , 1,
 dadosTeste['forma_envio_solicitacao'])
dadosTeste['forma_envio_solicitacao'] = np.where(dadosTeste['forma_envio_solicitacao'] == 'correio' , 2, da
dosTeste['forma_envio_solicitacao'])
print("Mapeamento de variáveis string para número feito com sucesso\n")
```

Código 6 - Mapeamento dos atributos formato string - parte 2

Após o mapeamento dos dados, utilizei a função *sample* da biblioteca *pandas* para aleatorizar os dados e retirar o índice das amostras, neste caso, utilizando a função *reset_index*, também da biblioteca *pandas*:

```
#--------------------RANDOMIZACAO DOS DADOS--------------------------------
```

```
dadosTreinamento = dadosTreinamento.sample(frac=1).reset_index(drop=True)
```

Código 7 - Randomização e remoção do índice

Com o objetivo de separar os atributos do rótulo, é necessário atribuir a duas variáveis separadas, uma lista dos atributos e uma lista dos rótulos, ambos ordenados:

Visto que os atributos são de ordem de grandeza diferentes, por exemplo, renda mensal e quantidade de dependentes, é necessário fazer uma normalização dos dados dos atributos, para isso, utilizei a função StandardScaler da biblioteca *sklearn.preprocessing*:

```
#--------------------DIVISAO DOS DADOS ENTRE VARIAVEIS E ROTULO-------------
```

```
y_treinamento = dadosTreinamento['inadimplente']
x_treinamento = dadosTreinamento.drop(['inadimplente'], axis=1)
```

Código 8 - Divisão dos atributos e dos rótulos

```
#--------------------NORMALIZACAO DO DADO-----------------------------------
escala = StandardScaler()
escala.fit(x_treinamento)

x_treinamento = escala.transform(x_treinamento)
x_teste_predicao = escala.transform(dadosTeste)
print("Normalização dos dados feita com sucesso\n")
```

Código 9 - Normalização dos atributos

A normalização dos dados foi a última etapa de tratamento dos dados, a etapa seguinte que desenvolvi foi a plotagem em gráfico histograma a fim de identificar o padrão dos dados, então relato que esta etapa, serve apenas para o estudo dos dados.

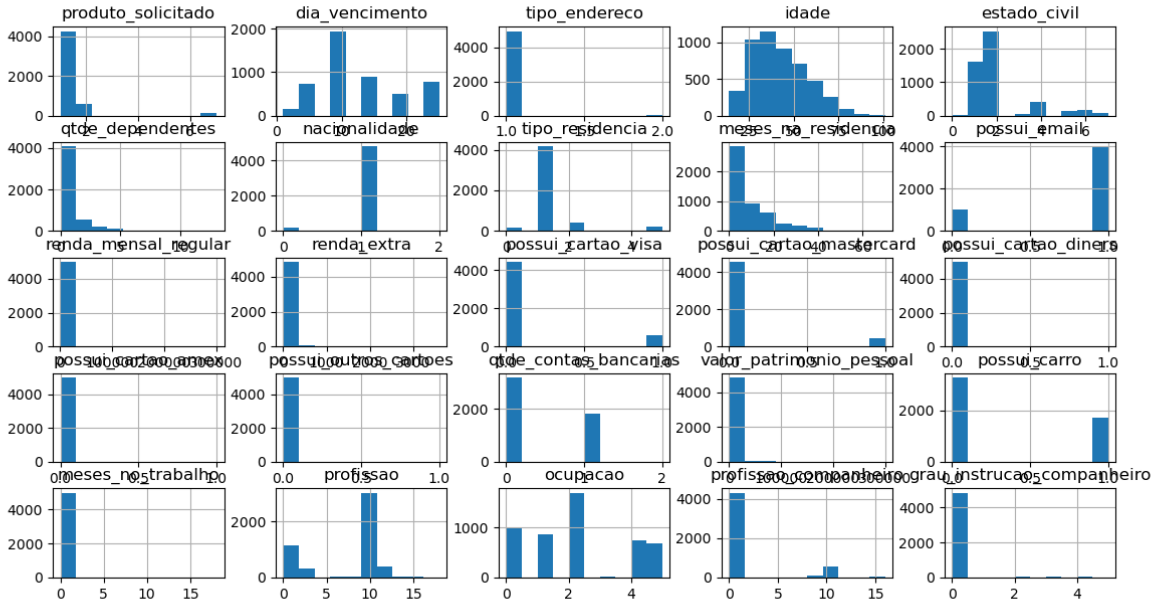

Figura 1 - Histograma dos dados

2.2 Treinamento

Para o treinamento, utilizei três classificadores diferentes: linear, logístico e KNN, ambos resultam em previsões diferentes dos dados então, foi feito uma mensuração da previsão utilizando a métrica R^2.

2.2.1 Classificador Linear

O classificador linear envolve as etapas de: instanciar o classificador, treinar o classificador utilizando método *fit* da função *LinearRegression* biblioteca *sklearn.linear_model* e utilizar o método *predict* para prever as respostas dos dados de treinamento. Após isso, utilizei a função *where* da biblioteca *numpy* para fazer um mapeamento da porcentagem de inadimplência para uma variável booleana.

Tal porcentagem foi um parâmetro a qual eu fui variando a fim obter o melhor resultado possível na plataforma do *Kaggle*. Por fim, para obter a métrica de resultado das previsões, utilizei o método *score*, que internamente utiliza o cálculo R^2 para avaliar os dados.

```
#--------------------REGRESSAO LINEAR -----------------------------------
```

```
regressor_linear= LinearRegression()
regressor_linear.fit(x_treinamento,y_treinamento)
y_prob_linear = regressor_linear.predict(x_teste_predicao)
y_predicao_linear = np.where(y_prob_linear > 0.484, 1, 0)
print('\tR2 da Regressão Linear:',regressor_linear.score(x_teste_predicao, y_predicao_linear),'\n\n')
```

Código 10 - Regressão linear

2.2.2 Classificador Logístico

O classificador logístico envolve as etapas de: instanciar o classificador, treinar o classificador utilizando método *fit* da função *LogisticRegression* biblioteca *sklearn.linear_model* e utilizar o método *predict_proba* para prever as respostas dos dados de treinamento. Após isso, utilizei a função *where* da biblioteca *numpy* para fazer um mapeamento da porcentagem de inadimplência para uma variável booleana.

Tal porcentagem foi um parâmetro a qual eu fui variando a fim obter o melhor resultado possível na plataforma do *Kaggle*. Por fim, para obter a métrica de resultado das previsões, utilizei o método *score*, que internamente utiliza o cálculo R^2 para avaliar os dados.

```
#--------------------REGRESSAO LOGISTICA -----------------------------------
regressor_logistico = LogisticRegression()
regressor_logistico.fit(x_treinamento,y_treinamento)
y_prob_logistic = regressor_logistico.predict_proba(x_teste_predicao)[:,1]
y_predicao_logistic = np.where(y_prob_logistic > 0.484, 1, 0)
print('\tR2 da Regressão Logística:',regressor_logistico.score(x_teste_predicao, y_predicao_logistic),'\n\n')
```

Código 11 - Regressão logística

2.2.3 Classificador KNN

O classificador KNN envolve as etapas de: instanciar o classificador, treinar o classificador utilizando método *fit* da função *KNeighborsClassifier* biblioteca *sklearn.neighbors* e utilizar o método *predict* para prever as respostas dos dados de treinamento. Após isso, utilizei a função *where* da biblioteca *numpy* para fazer um mapeamento da porcentagem de inadimplência para uma variável booleana.

Tal porcentagem foi um parâmetro a qual eu fui variando a fim obter o melhor resultado possível na plataforma do *Kaggle*. Por fim, para obter a métrica de resultado das previsões, utilizei o método *score*, que internamente utiliza a acurácia para avaliar os dados.

```
#-------------------REGRESSAO KNN COM N=20 ---------------------------------
regressor_knn= KNeighborsRegressor(n_neighbors=20,weights='distance')
regressor_knn.fit(x_treinamento,y_treinamento)
y_prob_knn = regressor_knn.predict(x_teste_predicao)
y_predicao_knn = np.where(y_prob_knn > 0.484, 1, 0)
print('\tAcurácia da Regressão KNN:',regressor_knn.score(x_teste_predicao, y_predicao_knn),'\n\n')
```

Código 12 - Regressão KNN

O classificador KNN possui como característica a quantidade de “vizinhos” que serão utilizados, para isso efetuei alguns testes em laço de repetição a fim de variar este número e ver o comportamento da acurácia. Porém, por fim, utilizei o número de 20 “vizinhos” visto que este era um número razoável para a classificação.

```
n: 16   Acurácia da Regressão KNN: 0.41150072754694167
n: 17   Acurácia da Regressão KNN: 0.40955057730539757
n: 18   Acurácia da Regressão KNN: 0.3977256904043681
n: 19   Acurácia da Regressão KNN: 0.3964745447964685
n: 20   Acurácia da Regressão KNN: 0.3892708107447863
n: 21   Acurácia da Regressão KNN: 0.38707861166220536
n: 22   Acurácia da Regressão KNN: 0.3798361149612347
n: 23   Acurácia da Regressão KNN: 0.3777328795849625
n: 24   Acurácia da Regressão KNN: 0.3693431826643023
```

Figura 2 - Laço de repetição do KNN

3.Resultados obtidos

Por fim, a última etapa do programa é a etapa de exportação dos dados para uma planilha em formato CSV que será interpretada pelo *Kaggle* e então avaliada, escolhi visto os resultados obtidos o classificador linear para submeter visto que foi analisado que este obteve o melhor *score* na plataforma.

```
#--------------------EXPORTACAO DA PREDICAO PARA CSV----------------

resposta = pd.DataFrame(index=dadosExemplo.index)
resposta['inadimplente'] = y_predicao_linear
resposta.to_csv('arquivo_resposta.csv')
print("Arquivo 'arquivo_resposta.csv' gerado com sucesso")
```

Código 13 - Exportação dos dados

Executando o programa, ele irá fazer a classificação utilizando os três classificadores detalhados neste relatório, onde segue abaixo as métricas obtidas:

Classificando utilizando regressor linear....

R2 da Regressão Linear: 0.2524635542136803

Classificando utilizando regressor logístico....

R2 da Regressão Logística: 0.941

Classificando utilizando regressor KNN com n=20....

Acurácia da Regressão KNN: 0.38929141149165836