Relatório 1 – Aprendizado de Máquina por Classificação

EEL891 – 2022.1 – Prof. Heraldo Almeida

Aluna: Martina Marques Jardim

DRE: 121078124

Id Kaggle: 10340340

1- Introdução:

O seguinte relatório tem como intuito descrever o código utilizado no primeiro trabalho da disciplina, sendo expostos os métodos de tratamento de dados, modelos preditivos utilizados e testados, bem como o motivo pelo qual o melhor modelo foi escolhido e os resultados intermediários. O objetivo desse projeto é criar um código que estima a inadimplência de um solicitante de crédito.

2 – Pré-processamento e tratamento de dados:

O *dataframe* “dados” refere-se à amostra de dados para treino (o arquivo ‘conjunto_de_treinamento.csv’)

```
# Modelo preditivo para aprovação de crédito #

import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelBinarizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

# Definindo os dados #

dados = pd.read_csv('conjunto_de_treinamento.csv')
dadosTeste = pd.read_csv('conjunto_de_teste.csv')
```

Inicialmente, foram definidas as variáveis categóricas, ou seja, aquelas que não são numéricas, podendo ser representadas por dados do tipo “object”, ou representadas por número simbólicos, como os códigos de telefone (DDD).

Em seguida, foram descobertas as cardinalidades de cada variável categórica. A cardinalidade consiste na diversidade de dados que estão relacionadas a cada variável.

```
# Descobrindo as variáveis categóricas, ajuda a escolher as variáveis úteis #

categoricas = [ x for x in dados.columns if dados[x].dtype == 'object' or x == 'codigo_area_telefone_residencial'
               or x == 'codigo_area_telefone_trabalho']

print (categoricas)

# Descobrindo a cardinalidade #

for variavel in categoricas:
    print ('\n%15s: '%variavel , "%4d categorias" % len(dados[variavel].unique()))
    print (dados[variavel].unique(), '\n')

# Serão excluídas as de cardinalidade muito altas, com resultados incoerentes e/ou repetitivas#

dados = dados.drop (['sexo', 'codigo_area_telefone_residencial', 'codigo_area_telefone_trabalho', 'possui_telefone_celula
                     'id_solicitante', 'estado_onde_nasceu', 'estado_onde_reside', 'estado_onde_trabalha', 'qtde_contas_b

dadosTeste = dadosTeste.drop (['sexo', 'codigo_area_telefone_residencial', 'codigo_area_telefone_trabalho', 'possui_telef
                    'id_solicitante', 'estado_onde_nasceu', 'estado_onde_reside', 'estado_onde_trabalha', 'qtde_contas_ba
```

As variáveis com resultados incoerentes, com cardinalidade muito altas e/ou informações repetitivas foram removidas.

As variáveis ‘sexo’, ‘possui telefone celular’ e são as que possuem resultados incoerentes, com quatro e uma categorias, respectivamente. A variável ‘estado_onde_nasceu’ também se encaixa nessa classificação, pois possui 28 categorias, número superior à quantidade de unidades federativas do Brasil.

As variáveis com cardinalidades muito altas são aquelas que possuem uma alta diversidade de dados. Isso afeta negativamente a previsão e o aprendizado do sistema, pois uma alta diversidade de dados em uma variável aumenta a quantidade de dimensões, o que dificulta o aprendizado da máquina. Nesse sentido, as variáveis são ‘id_solicitante’, ‘codigo_area_telefone_residencial’, ‘codigo_area_telefone_trabalho’.

As variáveis repetitivas são aquelas que se referem à mesma informação, ou seja, são aquelas que acrescentam informações que já foram fornecidas por outras variáveis. Nesta situação, tem-se ‘estado_onde_reside’ e ‘estado_onde_trabalha’, pois, as informações relativas a essas variáveis já foram informadas pelas variáveis ‘local_onde_reside’ e ‘local_onde_trabalha’.

A variável ‘qtde_contas_bancarias_especiais’ foi retirada pois, por inspeção, observou que o rendimento do aprendizado aumentou com sua eliminação.

```
# Binarizando
binarizador = LabelBinarizer()

for contador in ['possui_telefone_residencial', 'vinculo_formal_com_empresa', 'possui_telefone_trab
    dados [contador] = binarizador.fit_transform(dados[contador])
    dadosTeste [contador] = binarizador.fit_transform(dadosTeste[contador])

# One Hot Encoding para variaveis com mais de 3 categorias #

dados = pd.get_dummies(dados, columns = ['forma_envio_solicitacao'])
dadosTeste = pd.get_dummies(dadosTeste, columns = ['forma_envio_solicitacao'])
```

Pela inspeção da cardinalidade das variáveis, também é possível encontrar as variáveis binárias, ou seja, com apenas duas categorias. Dessa

forma, foi realizada a binarização dessas variáveis. Dessa forma, as duas categorias de ‘possui_telefone_residencial’, ‘vinculo_formal_com_empresa’ e ‘possui_telefone_trabalho’, foram codificadas em dois números (1 e 0).

Além disso, para a variável ‘forma_envio_solicitacao’, a qual possui três categorias codificadas por *strings*, foi aplicado o *One Hot Encoding*, a qual transforma essas três categorias em colunas do *dataframe*, com dados binários.

```
selecionados = ['produto_solicitado', 'dia_vencimento', 'tipo_endereco', 'idade',
       'estado_civil', 'qtde_dependentes', 'grau_instrucao', 'nacionalidade',
       'possui_telefone_residencial', 'tipo_residencia', 'meses_na_residencia',
       'possui_email', 'renda_mensal_regular', 'renda_extra',
       'possui_cartao_visa', 'possui_cartao_mastercard',
       'possui_cartao_diners', 'possui_cartao_amex', 'possui_outros_cartoes',
       'qtde_contas_bancarias',
       'valor_patrimonio_pessoal', 'possui_carro',
       'vinculo_formal_com_empresa', 'possui_telefone_trabalho',
       'meses_no_trabalho', 'profissao', 'ocupacao', 'profissao_companheiro',
       'grau_instrucao_companheiro', 'local_onde_reside',
       'local_onde_trabalha','forma_envio_solicitacao_correio', 'forma_envio_solicitacao_internet',
       'forma_envio_solicitacao_presencial']


alvo = 'inadimplente'

# Os espaços vazios e NotANumber serão substituídos por zero

dados = dados.fillna(0)
embaralhado = dados.sample(frac=1, random_state = 12345)
dadosTeste = dadosTeste.fillna(0)
```

Em seguida, é necessário tratar os últimos dados não numéricos presentes no *dataframe.* Esses são variáveis do tipo NaN (*Not a Number*) ou *strings* vazias. Isso é feito por meio da função *.fillna(0).* Esse procedimento é considerado o ideal pois, por esses dados não serem numéricos e por não trazerem nenhuma informação concreta. Aqui também é definida a variável alvo, que equivale a coluna ‘inadimplente’ do *dataframe.*

3 – Métodos preditivos

Foram aplicados dois métodos preditivos, sendo que, para ambos, foi utilizada a mesma base de dados, descrita na seção anterior. O código enviado contém somente a implementação do método *Random Forest,* uma vez que esse é o mais eficiente.

A variável alvo dessa modelo é a variável ‘inadimplente’.

O *dataframe* ‘embaralhado’ é o *dataframe* “dados” fora de ordem. O *dataframe* ‘embaralhado’ será utilizado para treinar ambos os algoritmos de previsão.

Inicialmente, aplicou-se o método de classificação por KNN.

```
dados = dados.fillna(0)
embaralhado = dados.sample(frac=1, random_state = 12345)
dadosTeste = dadosTeste.fillna(0)

matrizX = embaralhado.loc[:,embaralhado.columns!='inadimplente'].values
matrizY = embaralhado.loc[:,embaralhado.columns=='inadimplente'].values
matrizXTeste = dadosTeste.loc[:,dadosTeste.columns!='inadimplente'].values

scaler = MinMaxScaler()

linhasTreino = 20000

x_treino = matrizX[:linhasTreino,:]
y_treino = matrizY[:linhasTreino].ravel()
x_teste = matrizXTeste

x_treino = scaler.fit_transform(x_treino)
x_teste = scaler.fit_transform(x_teste)

classificador = KNeighborsClassifier(n_neighbors=40)

classificador = classificador.fit(x_treino, y_treino)

print ('resposta Treino:')

y_resposta_treino = classificador.predict(x_treino)

print (y_resposta_treino)

print ('resposta Teste:')

y_resposta_teste = classificador.predict(x_teste)

print (y_resposta_teste)
```

O *dataframe* “matrizXTeste” refere-se aos dados para teste. Com os resultados obtidos a partir desse conjunto, é gerado um documento *.csv*.

Em razão de o método de classificação por KNN se basear na distância entre duas amostras, é necessário colocar uma escala. Nesse sentido, foi escolhido o *MinMax Scaler.* Essa escala faz com que todos os valores do *dataframe* se encaixem entre 0 e 1. Essa característica verificou-se ser muito eficiente para o aprendizado de máquina, pois, além de aproximar as variáveis, também mantém a codificação do *dataframe,* ou seja, dados codificados pelos mesmos números continuarão a ser codificados pelo mesmo valor, contudo esse estará escalado. Por inspeção, verificou-se que essa foi a escala que trouxe a melhor acurácia. Ambos os *dataframes* (de treino e de teste) são escalados antes de ser aplicado o método preditivo.

A quantidade *k* de vizinhos utilizado para previsão também foi verificada por inspeção. Utilizando-se um laço de repetição *for* k variando em uma lista de índices de 10 a 43, encontrou-se que a melhor acurácia era obtida com k=40.

A variável “y_resposta_treino” refere-se às respostas da previsão aplicando-se o conjunto de treino embaralhado e escalado (*dataframe* ‘x_treino’) e a variável e “y_resposta_teste” refere-se às respostas da previsão aplicando-se o conjunto de treino embaralhado e escalado (*dataframe* ‘x_teste’).

A acurácia do modelo foi testada por meio de validação cruzada:

```
print ('AVALIAÇÃO DE RESULTADOS POR VALIDAÇÃO CRUZADA')
kfold  = KFold(n_splits=5, shuffle=False)
resultado = cross_val_score(classificador, x_treino, y_treino, cv = kfold, scoring = 'accuracy')
print(" Acurácia para cada bloco K-Fold : {0}".format(resultado))
print("Média das acurácias para Cross-Validation K-Fold: {0}".format(resultado.mean()))
```

Foi aplicado o método *K-Folds*, em que a quantidade de divisões é igual a cinco, e não será feito o embaralhamento dos dados, pois este já foi feito na definição do *dataframe* do treino.

Com as variáveis selecionadas e com o método do KNN, obteve-se o seguinte resultado:

```
resposta Treino:
[1 0 0 ... 0 0 1]
resposta Teste:
[1 1 0 ... 0 1 1]
AVALIAÇÃO DE RESULTADOS POR VALIDAÇÃO CRUZADA
 Acurácia para cada bloco K-Fold : [0.555   0.55425 0.56075 0.55
0.564  ]
Média das acurácias para Cross-Validation K-Fold: 0.5568

In [8]:
```

Com o intuito de melhorar o rendimento da classificação, foi aplicado o método da *Random Forest.*

```
floresta = RandomForestClassifier (criterion = 'gini', max_depth = 10,
                                   max_features = 31, min_samples_leaf = 8,
                                   min_samples_split = 10, random_state = 5)

floresta.fit(matrizX, matrizY)

y_resposta_treino = floresta.predict(matrizX)
y_resposta_teste = floresta.predict(matrizXTeste)
```

O classificador por Random Forest é representado pela variável “floresta”.

O critério escolhido para definir a Random Forest é a *Gini Impurity,* ou seja, a probabilidade de classificar incorretamente um conjunto de dados. *Max_depth* representa a profundida máxima da árvore. *Max_features* representa a máxima quantidade de variáveis a serem consideradas ao iniciar um novo nó. *Min_samples_leaf* representa a quantidade mínima de amostras que uma folha deve ter. *Min_samples_split* define a quantidade mínima de amostras que uma divisão deve ter.

Houve a tentativa de implementar a função *GridSearch.CV,* de forma a escolher os melhores parâmetros, contudo, está implementação deixou o código muito mais lento, além de sobrecarregar a máquina.

```
floresta = RandomForestClassifier ()

n_estimators = [100, 300, 500, 800, 1200]
max_depth = [5, 8, 15, 25, 30]
min_samples_split = [2, 5, 10, 15, 100]
min_samples_leaf = [1, 2, 5, 10]

hyperF = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split,
             min_samples_leaf = min_samples_leaf)

gridFloresta = GridSearchCV(floresta, hyperF, cv = 3, verbose = 1,
                      n_jobs = -1)

melhoresParametros = gridFloresta.fit(matrizX, matrizY)
```

Nesse sentido, aplicou-se os valores dos hiper parâmetros por inspeção, e encontrou-se:

```
floresta = RandomForestClassifier (criterion = 'gini', max_depth = 10,
                                   max_features = 31, min_samples_leaf = 8,
                                   min_samples_split = 10, random_state = 5)

floresta.fit(matrizX, matrizY)

y_resposta_treino = floresta.predict(matrizX)
y_resposta_teste = floresta.predict(matrizXTeste)
```

Ao aplicar a validação cruzada para a Random Forest, obtém-se o seguinte resultado:

```
AVALIAÇÃO DE RESULTADOS POR VALIDAÇÃO CRUZADA
A acurácia para cada bloco K-Fold: [0.58925 0.597   0.57875 0.58125
0.5875 ]
Média das acurácias para Cross-Validation K-Fold: 0.5867500000000001

In [19]:
```

Como é possível observar, houve um resultado melhor do que no método por KNN. Em razão disso, e em conformidade com o que foi dito no início da seção, o código com a implementação do método KNN foi descartado e apenas a implementação do Random Forest consta no código enviado.

4 – Conclusão

Em suma, nesse código é possível observar a aplicação de diversos conceitos de *Machine Learning,* bem como o uso de diversas funções em linguagem Python, de diferentes bibliotecas, principalmente *ScikitLearn*. Por meio dos procedimentos descritos acima, foi construído um modelo de aprendizado de máquina por classificação com o intuito de prever a inadimplência de um solicitante de crédito.

5 – Referências

Documentação sobre o classificador Random Forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Artigo sobre a *Gini Impurity:* https://towardsdatascience.com/gini-impurity-measure-dbd3878ead33#:~:text=Def%3A%20Gini%20Impurity%20tells%20us,lower%20the%20likelihood%20of%20misclassification.

Otimização de Hiper-parâmetros: https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6

Documentação dobre a função GridSearchCV: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Artigo sobre Validação Cruzada: https://drigols.medium.com/introdu%C3%A7%C3%A3o-a-valida%C3%A7%C3%A3o-cruzada-k-fold-2a6bced32a90