Introdução ao Aprendizado de Máquina

Relatório do Trabalho Prático 1 - EEL891 - Semestre 2023/2

Orientador: Heraldo Luis -- Aluno: Lucas Tavares Da Silva Ferreira

This document is the report of the first practical task of the discipline "Introduction to Machine Learning" (EEL891) at the Federal University of Rio de Janeiro (UFRJ). The work consists of building a classifier to support the credit approval decision.

The classifier to support the credit approval decision must identify among the customers who request a credit product (such as a credit card or a personal loan, for example) and who meet the essential prerequisites for credit approval, those that present a high risk of not being able to honor the payment, becoming defaulters.

Trabalho 1

Estudo das variáveis

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score

dados = pd.read_csv('conjunto_de_treinamento.csv', delimiter=',', decimal='.')
dados_teste = pd.read_csv('conjunto_de_teste.csv', delimiter=',', decimal='.')



dados['meses_na_residencia'] = dados['meses_na_residencia'].fillna(dados['meses_na_resid
encia'].median())

dados['profissao_companheiro'] = dados['profissao_companheiro'].fillna(dados['profissao_c
ompanheiro'].median())

dados['tipo_residencia'] = dados['tipo_residencia'].fillna(dados['tipo_residencia'].medi
an())

dados_teste['meses_na_residencia'] = dados_teste['meses_na_residencia'].fillna(dados_tes
te['meses_na_residencia'].median())

dados_teste['profissao_companheiro'] = dados_teste['profissao_companheiro'].fillna(dados_
teste['profissao_companheiro'].median())

dados_teste['tipo_residencia'] = dados_teste['tipo_residencia'].fillna(dados_teste['tipo
_residencia'].median())
```

Estados por região

In [ ]:

```
regioes = {'RJ':'sudeste',
           'ES':'sudeste',
```

```
          'MG':'sudeste',
          'SP':'sudeste',
          'MT':'centroOeste',
          'MS':'centroOeste',
          'DF':'centroOeste',
          'GO':'centroOeste',
          'RS':'sul',
          'PR':'sul',
          'SC':'sul',
          'SE':'nordeste',
          'AL':'nordeste',
          'BA':'nordeste',
          'PE':'nordeste',
          'RN':'nordeste',
          'PB':'nordeste',
          'MA':'nordeste',
          'PI':'nordeste',
          'CE':'nordeste',
          'AP':'norte',
          'RR':'norte',
          'PA':'norte',
          'RO':'norte',
          'TO':'norte',
          'AC':'norte',
          'AM':'norte',}

dados = dados.replace(regioes)
dados_teste = dados_teste.replace(regioes)
```

Preenchimento de nulos, mapeamento de categorias...

Verificando o valor médio de cada atributo em cada classe

In [ ]:

```
#Lendo a quantidade de amostras de cada classe

print(dados['inadimplente'].value_counts())

print(dados.groupby(['inadimplente']).mean().T)
```

Preenchendo os espaços nulos

In [ ]:

```
dados = dados.fillna(0)
dados_teste  = dados_teste.fillna(0)
```

Leitura de variáveis categóricas

In [ ]:

```
#Imprimir os tipos de dados de cada coluna
variaveis_categoricas = [ x for x in dados.columns if dados[x].dtype == 'object']

#print(dados.dtypes)
#print(variaveis_categoricas)
```

In [ ]:

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

Eliminando variáveis

In [ ]:

In [ ]:

```
#Descartando dados de alta cardinalidade
dados = dados.drop (['codigo_area_telefone_residencial', 'codigo_area_telefone_trabalho',
'possui_telefone_celular',
                     'id_solicitante','estado_onde_trabalha','qtde_contas_bancarias_espe
ciais','estado_onde_nasceu','grau_instrucao_companheiro','local_onde_trabalha', 'grau_ins
trucao'], axis = 1)


#dados = dados.drop(['meses_no_trabalho'],axis=1)
#dados = dados.drop(['forma_envio_solicitacao'],axis=1)
dados_teste = dados_teste.drop (['codigo_area_telefone_residencial', 'codigo_area_telefon
e_trabalho', 'possui_telefone_celular',
                     'id_solicitante','estado_onde_trabalha','estado_onde_nasceu','qtde_
contas_bancarias_especiais','grau_instrucao_companheiro','local_onde_trabalha','grau_inst
rucao'], axis = 1)


#dados_teste = dados_teste.drop(['meses_no_trabalho'],axis=1)
#dados_teste = dados_teste.drop(['estado_onde_reside'],axis=1)
#dados_teste = dados_teste.drop(['forma_envio_solicitacao'],axis=1)
```

Aplicando o One-Hot-Encoding

In [ ]:

```
#colocando 0 onde há espaço em branco
dados = pd.get_dummies(dados,columns = ['forma_envio_solicitacao'])
dados = pd.get_dummies(dados,columns = ['estado_onde_reside'])
dados = pd.get_dummies(dados,columns = ['sexo'])
#dados = pd.get_dummies(dados,columns = ['produto_solicitado'])
#dados = pd.get_dummies(dados,columns = ['ocupacao'])
#dados = pd.get_dummies(dados,columns = ['tipo_residencia'])
#dados = pd.get_dummies(dados,columns = ['local_onde_reside'])


dados_teste = pd.get_dummies(dados_teste,columns = ['forma_envio_solicitacao'])
dados_teste = pd.get_dummies(dados_teste,columns = ['estado_onde_reside'])
dados_teste = pd.get_dummies(dados_teste,columns = ['sexo'])
#dados_teste = pd.get_dummies(dados_teste,columns = ['produto_solicitado'])
#dados_teste = pd.get_dummies(dados_teste,columns = ['ocupacao'])
#dados_teste = pd.get_dummies(dados_teste,columns = ['tipo_residencia'])
#dados_teste = pd.get_dummies(dados_teste,columns = ['local_onde_reside'])
```

In [ ]:

```
#Binarizando variáveis
binarizador = LabelBinarizer()
for x in ['vinculo_formal_com_empresa', 'possui_telefone_trabalho','possui_telefone_resid
encial']:
  dados[x] = binarizador.fit_transform(dados[x])
  dados_teste[x] = binarizador.fit_transform(dados_teste[x])
#print(dados.head(10).T)
```

In [ ]:

```
#display(dados.columns)
```

In [ ]:

```
dados.columns.tolist()
```

Selecionando atributos

In [ ]:

```
atributosSelecionados = [
 'id_solicitante',
 'produto_solicitado',
```

```
 'dia_vencimento',
 'forma_envio_solicitacao',
 'tipo_endereco',
 'sexo',
 'idade',
 'estado_civil',
 'qtde_dependentes',
 'grau_instrucao',
 'nacionalidade',
 'estado_onde_nasceu',
 'estado_onde_reside',
 'possui_telefone_residencial',
 'codigo_area_telefone_residencial',
 'tipo_residencia',
 'meses_na_residencia',
 'possui_telefone_celular',
 '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',
 'qtde_contas_bancarias_especiais',
 'valor_patrimonio_pessoal',
 'possui_carro',
 'vinculo_formal_com_empresa',
 'estado_onde_trabalha',
 'possui_telefone_trabalho',
 'codigo_area_telefone_trabalho',
 'meses_no_trabalho',
 'profissao',
 'ocupacao',
 'profissao_companheiro',
 'local_onde_reside',
 'local_onde_trabalha',
 ]

alvo = 'inadimplente'
```

Permutação Aleatória de dados

In [ ]:

```
dados_embaralhados = dados.sample(frac=1,random_state=12345)
dados_teste_embaralhados =dados_teste.sample(frac=1,random_state=12)
```

In [ ]:

```
x = dados_embaralhados.loc[:,dados_embaralhados.columns != 'inadimplente'].values
y = dados_embaralhados.loc[:,dados_embaralhados.columns == 'inadimplente'].values

x_testagem = dados_teste.loc[:,dados_teste.columns != 'inadimplente'].values
y_testagem = dados_teste.loc[:,dados_teste.columns == 'inadimplente'].values
```

Seleção de amostras para calibragem do algoritmo KNN e Logistic Regression

In [ ]:

```
#treino
x_treino = x[:17000,:-1]
y_treino= y[:17000,-1].ravel()

#teste
x_teste = x[17000:,:-1]
```

```
y_teste = y[17000:,-1].ravel()
```

Scaler para calibragem do algoritmo KNN e Logistic Regression

In [ ]:

```
scaler = MinMaxScaler()
scaler.fit_transform(x_treino)
x_treino = scaler.transform(x_treino)
x_teste = scaler.transform(x_teste)
```

In [ ]:

```
#from sklearn.preprocessing import StandardScaler

#StdSc = StandardScaler()
#StdSc = StdSc.fit(x_treino)
#x_treino = StdSc.transform(x_treino)
#x_teste = StdSc.transform(x_teste)
```

Seleção de Amostras para geração de CSV

In [ ]:

```
#treino
x_treino_final = x
y_treino_final = y.ravel()

#teste
x_teste_final = x_testagem
```

Método KNN

In [ ]:

```
for k in range(1,4): #colocado um alcance curto para que a execução de todo o código, na
hora de ser avaliado, não demorar muito

    classificador = KNeighborsClassifier(n_neighbors=k)
    classificador = classificador.fit(x_treino,y_treino)

    y_resposta_treino = classificador.predict(x_treino)
    y_resposta_teste  = classificador.predict(x_teste)

    acuracia_treino = sum(y_resposta_treino==y_treino)/len(y_treino)
    acuracia_teste  = sum(y_resposta_teste ==y_teste) /len(y_teste)

    print(
        "%3d"%k,
        "%6.1f" % (100*acuracia_treino),
        "%6.1f" % (100*acuracia_teste)
        )
#    scores = cross_val_score(classificador, x,y.ravel(), cv = 5)
#    print('k = %2d'%k, 'scores = ', scores, 'acuracia media = %6.1f' % (100*sum(scores)/
5))
```

Geração de CSV (KNN)

In [ ]:

```
classificador = KNeighborsClassifier(n_neighbors=32)
classificador = classificador.fit(x_treino_final,y_treino_final)
y_resposta_treino = classificador.predict(x_treino_final)
y_resposta_testeKNN = classificador.predict(x_teste_final)
```

In [ ]:

```
#aux = pd.read_csv('conjunto_de_teste.csv')
#resposta_KNN = pd.DataFrame({'id_solicitante':aux.pop('id_solicitante'), 'inadimplente':
np.squeeze(np.int16(y_resposta_testeKNN ))})
#resposta_KNN.to_csv("resposta_KNN.csv",index=False)
#Como o random forest obteve melhores resultados, decidi não gerar o CSV deste método
```

Método Logistic Regression

In [ ]:

```
from sklearn.linear_model import LogisticRegression

for k in range(-1,10):


    c = pow(10,k)
    classificador = LogisticRegression(penalty = 'l2', C = c)
    classificador = classificador.fit(x_treino,y_treino)

    y_resposta_treino = classificador.predict(x_treino)
    y_resposta_teste  = classificador.predict(x_teste)

    acuracia_treino = sum(y_resposta_treino==y_treino)/len(y_treino)
    acuracia_teste  = sum(y_resposta_teste ==y_teste) /len(y_teste)

    print(
        "%14.6f"%c,
        "%6.1f" % (100*acuracia_treino),
        "%6.1f" % (100*acuracia_teste)
        )
```

Geração de CSV (Logistic Regression)

In [ ]:

```
c = 1
classificador = LogisticRegression(penalty = 'l2', C = c)
classificador = classificador.fit(x_treino_final,y_treino_final)
y_resposta_treino = classificador.predict(x_treino_final)
y_resposta_testeRegressaoLogistica = classificador.predict(x_teste_final)
```

In [ ]:

```
#aux = pd.read_csv('conjunto_de_teste.csv')
#resposta_Logistic6 = pd.DataFrame({'id_solicitante':aux.pop('id_solicitante'), 'inadimpl
ente':np.squeeze(np.int16(y_resposta_testeRegressaoLogistica ))})
#resposta_Logistic6.to_csv("resposta_Logistic6.csv",index=False)
#Como o random forest obteve melhores resultados, decidi não gerar o CSV deste método
```

Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import LeaveOneOut
import numpy as np
import math
import time
```

```
import warnings

x = dados_embaralhados.loc[:,dados_embaralhados.columns != 'inadimplente'].values #redefi
nindo x e y
y = dados_embaralhados.loc[:,dados_embaralhados.columns == 'inadimplente'].values

x_treino = x
y_treino = y.ravel()
```

Laço de repetição para simples testagem de variação de número de árvores

In [ ]:

```
for n in range(1,11):

  model = RandomForestClassifier(criterion='gini', max_depth=15, max_features=7, min_sam
ples_leaf=5, min_samples_split=0.001, n_estimators= n)
  model.fit(x_treino,y_treino)

  kfold  = KFold(n_splits=10, shuffle=True)
  resultado = cross_val_score(model, x_treino, y_treino, cv = kfold, scoring='accuracy')
  print("n = %i"%n)
  print("K-Fold Scores: {0}".format(resultado))
  print("Media da acuracia por validação cruzada K-Fold: {0}".format(resultado.mean()))
  #model.score(x_teste,y_teste)
  #print('n_estimators = %.i '%k, model.score(x_teste,y_teste))


#scores = cross_val_score(classificador, x,y.ravel(), cv = 5)
#print('k = %2d'%k, 'scores = ', scores, 'acuracia media = %6.1f' % (100*sum(scores)/5))
```

Testando variáveis específicas

In [ ]:

```
model = RandomForestClassifier(criterion='gini', max_depth=15, max_features=7, min_sampl
es_leaf=5, min_samples_split=0.001, n_estimators= 200)
model.fit(x_treino,y_treino)
kfold  = KFold(n_splits=10, shuffle=True)
resultado = cross_val_score(model, x_treino, y_treino, cv = kfold, scoring='accuracy')
print("K-Fold Scores: {0}".format(resultado))
print("Media de acuracia por validação cruzada K-Fold: {0}".format(resultado.mean()))
#model.score(x_teste,y_teste)
```

Geração de CSV (Random Forest)

In [ ]:

```
model = RandomForestClassifier(criterion='gini', max_depth=15, max_features=7, min_sampl
es_leaf=5, min_samples_split=0.001, n_estimators= 200)
model.fit(x_treino_final,y_treino_final)
#model.score(x_teste_final,y_teste_final)
```

In [ ]:

```
y_random_forest = model.predict(x_teste_final)
```

In [ ]:

```
print(y_random_forest)
```

Salvando o CSV gerado

In [ ]:

```
#y_random_forest = clf.predict(x_teste)
aux = pd.read_csv('conjunto_de_teste.csv')
resposta_random_forestClassificador_teste5 = pd.DataFrame({'id_solicitante':aux.pop('id_solicitante'), 'inadimplente':np.squeeze(np.int16(y_random_forest))})
resposta_random_forestClassificador_teste5.to_csv("resposta_random_forestClassificador_teste5.csv",index=False)
```

Conclusão

O random forest obteve os melhores resultados ao comparar o CSV gerado por esse método com o CSV das respostas, com uma média de acurácia maior do que os outros métodos (0.58)