aula_02

June 18, 2020

[1]:
```
''' Paulo Simplício Braga
        15.06.2020
'''

import sys
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm
import seaborn as sn
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn import preprocessing
from warnings import simplefilter
# Ignora 'future warnings'
simplefilter(action='ignore', category=FutureWarning)


def create_data_frame(path):
        data = pd.read_csv(path)
        # As linhas abaixo separam a feature 'quality' em duas classes (baixa␣
 ↪qualidade e alta qualidade):
        # se quality<=6, quality = 0 || se quality >= 7, quality 1
        data['quality'] = np.where(data['quality'] <= 6, 0, data['quality'] )
        data['quality'] = np.where(data['quality'] >= 7, 1, data['quality'] )


        return data
```

```
def smote(data, X, y):
        # SMOTE (Synthetic Minority Oversampling Technique):
        # O SMOTE funciona à partir da criação de amostras da classe␣
↪minoritária, ou seja, que está em
        # menor número no dataset (neste exemplo, 'alta qualidade'). Este é um␣
↪tipo de aumento de dados
        # para a classe minoritária que está desequilibrada em relação a classe␣
↪majoritária

        print("Quantidade de baixa qualidade antes do SMOTE:",␣
↪len(y[y['quality']==0]))
        print("Quantidade de alta qualidade antes do SMOTE:",␣
↪len(y[y['quality']==1]))

        # Aqui é criado o objeto 'os' que é do tipo SMOTE
        os = SMOTE(random_state=0)
        # A função 'train_test_split()', retorna uma lista de treinamento/teste␣
↪para x e y
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
↪2, shuffle = True,\

↪    stratify = y)
        columns = X_train.columns

        os_data_X, os_data_y = os.fit_sample(X_train, y_train)
        os_data_X = pd.DataFrame(data=os_data_X, columns=columns)
        os_data_y = pd.DataFrame(data=os_data_y, columns=['quality'])

        print("Quantidade de baixa qualidade após SMOTE:",␣
↪len(os_data_y[os_data_y['quality']==0]))
        print("Quantidade de alta qualidade após SMOTE:",␣
↪len(os_data_y[os_data_y['quality']==1]))

        return os_data_X, os_data_y

def rfe(data, os_data_X, os_data_y, size = 11):
        # RFE (Recursive Feature Elimination)
        # O RFE é baseado na ideia de repetir a construção de um modelo␣
↪inúmeras vezes
        # e escolher entre o feature que se sai melhor (ou pior)

        logreg = LogisticRegression(max_iter=1000) # Criado objeto da Regressão␣
↪Logística

        rfe = RFE(logreg, size)         # Criado o objeto do RFE, propriamente␣
↪dito. Importante notar que aqui é passado
```

```
                                                        # o objeto 'logreg'␣
↪criado anteriormente, juntamente com a quantidade final de
                                                        # features que o␣
↪algorítmo deve gerar (size)
        rfe = rfe.fit(os_data_X, os_data_y.values.ravel()) # Por fim, o␣
↪algorítmo RFE é iníciado com os valores de

↪  # treinamento os_data_X e "os_data_y.values.ravel()". O

↪  # método numpy ravel() é passado para fazer da array

↪  # uma lista (ou achatá-la)
        # Rotina para pegar automaticamente os valores 'True' e linkar com os␣
↪features da 'os_data_X'
        count = 0
        count_array = 0
        rfe_feature_array = [0]*size # Array para guardar os resultados 'True'␣
↪da RFE
        for i in rfe.support_:
                if i == True:
                        rfe_feature_array[count_array] = os_data_X.
↪columns[count]
                        count_array+=1
                count+=1
        return rfe_feature_array

# Opcional: faz o teste do modelo e verifica os features
def check_model(X, y, cols):
        # Implementação do modelo
        X=X[cols]
        y=y['quality']

        logit_model=sm.Logit(y,X)
        result=logit_model.fit()
        print(result.summary2())

def apply_logistic_regression(X, y, test_s):
        # Logistic Regression Model Fitting
        # Na linha abaixo, o dataframe é dividido na porporção␣
↪teste_size=test_s (0.2, neste exemplo) e
        # train_size=total_size-test_size (0.8, neste exemplo). A flag 'shuffle␣
↪= True' faz o embaralhamento
        # dos dados. Já a flag 'stratify = y' mantém a proporcionalidade das␣
↪variáveis
        X_train, X_test, y_train, y_test = train_test_split(X, y,␣
↪test_size=test_s,\
```

```
                                                                        ␣
↪shuffle = True, stratify = y)
        logreg = LogisticRegression(max_iter=1000) # Objeto logreg criado.␣
↪max_iter = 1000 para não exceder o número máximo iterações

        y_train = y_train.values.ravel() # Para achatar a coluna 'quality' e␣
↪tranformá-la em uma matriz 1d

        logreg.fit(X_train, y_train) # Aqui é feito o treinamento com os␣
↪datasets de treinamento de X e y

        y_pred = logreg.predict(X_test) # Por fim, a predição é feita e salva␣
↪na variável y_pred e a acurácia é
                                                                # então␣
↪impressa na linha seguinte
        print('\nAcurácia do classificador de Regressão Logística no test set: \
                {:.2f}'.format(logreg.score(X_test, y_test)),)

        print('Acurácia do classificador de Regressão Logística no train set: \
                {:.2f}'.format(logreg.score(X_train, y_train)), '\n')
        return y_pred, y_test


def print_matriz_de_conf(y_pred, y_test):
        # Aplicando a matriz de confusão com os valores previstos e os valores␣
↪do dataset de test
        c_m = confusion_matrix(y_test, y_pred, labels=[1,0])

        # Considerando verdadeiro positivo como alta qualidade (1) e verdadeiro␣
↪negativo como
        # baixa qualidade (0)
        verdadeiro_positivo = c_m[0][0]
        falso_positivo = c_m[0][1]
        falso_negativo = c_m[1][0]
        verdadeiro_negativo = c_m[1][1]
        soma = 0
        for i in range (2):
                for j in range(2):
                        soma += c_m[i][j]

        print("VP", verdadeiro_positivo)
        print("FP", falso_positivo)
        print("FN", falso_negativo)
        print("VN", verdadeiro_negativo, '\n')
```

```
        print("Acurácia = ", "{:.2f}".
 ↪format((verdadeiro_positivo+verdadeiro_negativo)/soma))
        print("Sensibilidade = ","{:.2f}".format(verdadeiro_positivo/
 ↪(verdadeiro_positivo + falso_negativo)))
        print("Especificidade = ","{:.2f}".format(verdadeiro_negativo/
 ↪(verdadeiro_negativo + falso_positivo)),'\n')

# Opcional: aplica a matriz de correlação:
def correlation_matrix(df):
        # print(df.corr())
        sn.heatmap(df.corr(), annot=True)
        plt.show()

def add_separator(val, sep = "=",):
        for i in range(val):
                print(sep, end='')
        print(flush=True)

def main():
        path = '../Data/winequality-red.csv'
        data = create_data_frame(path)

        X = data.loc[:, data.columns != 'quality'] # X recebe todas as colunas,␣
 ↪exceto a coluna 'quality'
        y = data.loc[:, data.columns == 'quality'] # y recebe a coluna 'quality'

        print ("Regressão Logística sem chamar os algorítmos SMOTE e RFE:")
        # A linha abaixo chama a função de Regressão Logística com test_size =␣
 ↪0.2. O train_size será,
        # automaticamente, 0.8
        y_pred, y_test = apply_logistic_regression(X, y, 0.2)
        print_matriz_de_conf(y_pred, y_test)

        add_separator(55, "_")

        print ("\nRegressão Logística chamando os algorítmos SMOTE e RFE:\n")
        os_data_X, os_data_y = smote(data, X, y)

        cols = rfe(data, os_data_X, os_data_y, 6)

        y_pred, y_test = apply_logistic_regression(os_data_X, os_data_y, 0.2)

        print_matriz_de_conf(y_pred, y_test)

if __name__ == '__main__':
    main()
```

```
Regressão Logística sem chamar os algorítmos SMOTE e RFE:

Acurácia do classificador de Regressão Logística no test set:            0.87
Acurácia do classificador de Regressão Logística no train set:           0.88

VP 11
FP 32
FN 9
VN 268

Acurácia =  0.87
Sensibilidade =  0.55
Especificidade =  0.89

-----------------------------------------------------

Regressão Logística chamando os algorítmos SMOTE e RFE:

Quantidade de baixa qualidade antes do SMOTE: 1382
Quantidade de alta qualidade antes do SMOTE: 217
Quantidade de baixa qualidade após SMOTE: 1105
Quantidade de alta qualidade após SMOTE: 1105

Acurácia do classificador de Regressão Logística no test set:            0.83
Acurácia do classificador de Regressão Logística no train set:           0.81

VP 194
FP 27
FN 46
VN 175

Acurácia =  0.83
Sensibilidade =  0.81
Especificidade =  0.87
```

[ ]:

[ ]: