TESTE TÉCNICO DATA SCIENCE

2) Modelagem - Redes Neurais

Rômulo Róseo Rebouças

```
import random
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from numpy.random import seed
from tensorflow.random import set_seed
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Model, Sequential

import numpy as np
import pandas as pd
import matplotlib
```

RNN com base de dados sem tratamento de balanceamento

```
##-- Leitua Cadastro sem balanceamento
df = pd.read_pickle('bd_SemBalanceamento.pkl')
df
```

	price	minimum_nights	number_of_reviews	calculated_host_listings_count	availability_365	neighbourhood_cod	room_type
0	221	5	260	1	304	32	
1	307	3	85	1	10	62	
2	160	7	238	11	328	32	
3	273	2	181	1	207	62	
4	135	3	353	1	101	32	
...	...	...	...	...	...	...	
26609	763	1	0	61	327	62	
26610	94	1	0	4	180	109	
26611	141	1	0	1	365	28	
26613	160	5	0	3	269	132	

```
####
print(df['room_type_tgt'].unique())
print(df.shape)

rotulos = np.array(df['room_type_tgt'])
features = np.array(df.iloc[:, 0:-1])
```

```
[0 2 3 1]
(23845, 7)
```

```
##-- Separando os dados:

perc_train = 0.7

n_train = int(features.shape[0]*perc_train)
n_test = int(features.shape[0]*(1-perc_train))

x_train =  features[0:n_train,:]
y_train = rotulos[0:n_train]
```

```
x_test =  features[0:n_test,:]
y_test = rotulos[0:n_test]

# transformar categorias em one-hot-encoding
y_train = keras.utils.to_categorical(y_train, 4)
y_test = keras.utils.to_categorical(y_test, 4)

print("\nConj. Train: ", len(x_train))
print("Conj. Y Train : ", len(x_train))
print("\nConj. Test ", len(x_test))
print("Conj. Y Test : ", len(y_test))
```

```
Conj. Train:  16691
Conj. Y Train :  16691

Conj. Test  7153
Conj. Y Test :  7153
```

```
##-- Rede neural profunda
def model_rnn (input_shape, dropout_rate=0.0):

    inputs = keras.Input(shape=input_shape)
    x = layers.BatchNormalization()(inputs)
    x = layers.Dense(32, activation="relu")(x)
    x = layers.Dense(64, activation="relu")(x)
    x = layers.Dropout(dropout_rate)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Dense(64, activation="relu")(x)
    x = layers.Dense(32, activation="relu")(x)
    outputs = layers.Dense(4, activation="softmax")(x)

    return keras.Model(inputs, outputs)
```

```
input_shape = 6
```

```
model = model_rnn(input_shape , 0.2)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 6)]               0
_________________________________________________________________
batch_normalization (BatchNo (None, 6)                 24
_________________________________________________________________
dense (Dense)                (None, 32)                224
_________________________________________________________________
dense_1 (Dense)              (None, 64)                2112
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
batch_normalization_1 (Batch (None, 64)                256
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 132
=================================================================
Total params: 8,988
Trainable params: 8,848
Non-trainable params: 140
_________________________________________________________________
```

```
##-- inicializando e treinando
##-- Taxa de aprendizado inicial de 0.001 e com decaimento em todas as épocas exponencial a -0.3
seed(1)
set_seed(2)

def scheduler(epoch, lr):
    return np.clip(lr * tf.math.exp(-0.3), 0.00001, 0.001)

callbacklr = tf.keras.callbacks.LearningRateScheduler(scheduler)
```

```
##-- Uso de pesos para as classes: menor peso para classes majoritárias (0 e 2) e maior peso para classes minoritárias (1 e 3)
class_weight = {0: 0.5, 1: 0.7, 2: 0.5, 3: 0.9}

epochs = 20
batch_size = 16

model.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adam(lr=0.001),
              metrics=[tf.keras.metrics.Precision(name='precision'), tf.keras.metrics.Recall(name='recall'), 'accuracy'] )
##-- Conjunto
hist_bdnormal = model.fit(x_train, y_train, class_weight=class_weight,
             callbacks=[callbacklr], batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Epoch 1/20
1044/1044 [==============================] - 6s 4ms/step - loss: 0.3764 - precision: 0.7509 - recall: 0.6508 - accuracy: 0.7258
Epoch 2/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3351 - precision: 0.7541 - recall: 0.6929 - accuracy: 0.7346
Epoch 3/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3229 - precision: 0.7693 - recall: 0.7131 - accuracy: 0.7526
Epoch 4/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3147 - precision: 0.7753 - recall: 0.7285 - accuracy: 0.7591
Epoch 5/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3247 - precision: 0.7699 - recall: 0.7171 - accuracy: 0.7549
Epoch 6/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3119 - precision: 0.7774 - recall: 0.7292 - accuracy: 0.7581
Epoch 7/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3069 - precision: 0.7792 - recall: 0.7345 - accuracy: 0.7627
Epoch 8/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3137 - precision: 0.7723 - recall: 0.7244 - accuracy: 0.7558
Epoch 9/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3186 - precision: 0.7720 - recall: 0.7251 - accuracy: 0.7574
Epoch 10/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3207 - precision: 0.7689 - recall: 0.7224 - accuracy: 0.7547
Epoch 11/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3174 - precision: 0.7713 - recall: 0.7215 - accuracy: 0.7530
Epoch 12/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3114 - precision: 0.7741 - recall: 0.7316 - accuracy: 0.7584
Epoch 13/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3121 - precision: 0.7823 - recall: 0.7334 - accuracy: 0.7631
Epoch 14/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3133 - precision: 0.7721 - recall: 0.7218 - accuracy: 0.7544
```

```
Epoch 15/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3124 - precision: 0.7746 - recall: 0.7285 - accuracy: 0.7576
Epoch 16/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3110 - precision: 0.7757 - recall: 0.7272 - accuracy: 0.7582
Epoch 17/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3141 - precision: 0.7798 - recall: 0.7306 - accuracy: 0.7621
Epoch 18/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3167 - precision: 0.7695 - recall: 0.7239 - accuracy: 0.7526
Epoch 19/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3198 - precision: 0.7713 - recall: 0.7247 - accuracy: 0.7523
Epoch 20/20
1044/1044 [==============================] - 4s 4ms/step - loss: 0.3113 - precision: 0.7765 - recall: 0.7295 - accuracy: 0.7587
```

```
hist_bdnormal_t = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=0)

score1_Tr = model.evaluate(x_train, y_train, verbose = 0)
score1_Te = model.evaluate(x_test, y_test, verbose = 0)
```

RNN com Balanceamento SMOTEENN (combinado)

```
##-- Leitua Cadastro com balanceamento - SMOTEENN (combinado)
df = pd.read_pickle('bd_SMOTEENN_Comb.pkl')
df
```

	price	minimum_nights	number_of_reviews	calculated_host_listings_count	availability_365	neighbourhood_cod	room_
0	-0.439785	0.025927	7.947134	-0.233806	0.629464	-0.569045	
1	-0.089480	-0.079835	2.309906	-0.233806	-1.457292	0.264563	
2	-0.790090	-0.079835	10.942918	-0.233806	-0.811392	-0.569045	
3	0.631497	-0.079835	0.151653	-0.205984	1.062430	-0.569045	
4	0.859602	-0.026954	-0.299325	-0.205984	-1.521172	-0.569045	
...	...	...	...	...	...	...	
16715	-0.689149	-0.185597	-0.428176	-0.168618	1.062430	0.156860	
16716	1.062020	0.184068	0.091574	0.069292	0.622966	1.064389	

```
####
print(df['room_type_tgt'].unique())
print(df.shape)

rotulos = np.array(df['room_type_tgt'])
features = np.array(df.iloc[:, 0:-1])
```

```
[0 1 2 3]
(16720, 7)
```

```
##-- Separando os dados:

perc_train = 0.7

n_train = int(features.shape[0]*perc_train)
n_test = int(features.shape[0]*(1-perc_train))

x_train =  features[0:n_train,:]
y_train = rotulos[0:n_train]

x_test =  features[0:n_test,:]
y_test = rotulos[0:n_test]
```

```
# transformar categorias em one-hot-encoding
y_train = keras.utils.to_categorical(y_train, 4)
y_test = keras.utils.to_categorical(y_test, 4)

print("\nConj. Train: ", len(x_train))
print("Conj. Y Train : ", len(x_train))
print("\nConj. Test ", len(x_test))
print("Conj. Y Test : ", len(y_test))
```

```
Conj. Train:  11704
Conj. Y Train :  11704

Conj. Test  5016
Conj. Y Test :  5016
```

```
input_shape = 6

model_bdBal = model_rnn(input_shape , 0.2)
model_bdBal.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 6)]               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 6)                 24
_________________________________________________________________
dense_5 (Dense)              (None, 32)                224
_________________________________________________________________
dense_6 (Dense)              (None, 64)                2112
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
batch_normalization_3 (Batch (None, 64)                256
_________________________________________________________________
dense_7 (Dense)              (None, 64)                4160
_________________________________________________________________
```

```
dense_8 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_9 (Dense)              (None, 4)                 132
=================================================================
Total params: 8,988
Trainable params: 8,848
Non-trainable params: 140
_________________________________________________________________
```

```
##-- inicializando e treinando
##-- Taxa de aprendizado inicial de 0.001 e com decaimento em todas as épocas exponencial a -0.3
seed(1)
set_seed(2)

def scheduler(epoch, lr):
    return np.clip(lr * tf.math.exp(-0.3), 0.00001, 0.001)

callbacklr = tf.keras.callbacks.LearningRateScheduler(scheduler)

##-- Uso de pesos para as classes: menor peso para classes majoritárias (0 e 2) e maior peso para classes minoritárias (1 e 3)
class_weight = {0: 0.5, 1: 0.7, 2: 0.5, 3: 0.9}

epochs = 20
batch_size = 16

model_bdBal.compile(loss='categorical_crossentropy', optimizer=keras.optimizers.Adam(lr=0.001),
              metrics=[tf.keras.metrics.Precision(name='precision'), tf.keras.metrics.Recall(name='recall'), 'accuracy'] )
##-- Conjunto
hist_bdBal = model_bdBal.fit(x_train, y_train, class_weight=class_weight,
             callbacks=[callbacklr], batch_size=batch_size, epochs=epochs, verbose=1)
```

```
Epoch 1/20
732/732 [==============================] - 4s 4ms/step - loss: 0.3101 - precision: 0.8319 - recall: 0.7276 - accuracy: 0.7953
Epoch 2/20
732/732 [==============================] - 3s 4ms/step - loss: 0.2041 - precision: 0.8800 - recall: 0.8603 - accuracy: 0.8699
Epoch 3/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1829 - precision: 0.8987 - recall: 0.8831 - accuracy: 0.8916
Epoch 4/20
```

```
732/732 [==============================] - 3s 4ms/step - loss: 0.1746 - precision: 0.8961 - recall: 0.8808 - accuracy: 0.8893
Epoch 5/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1777 - precision: 0.8936 - recall: 0.8782 - accuracy: 0.8847
Epoch 6/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1799 - precision: 0.8952 - recall: 0.8798 - accuracy: 0.8875
Epoch 7/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1782 - precision: 0.8982 - recall: 0.8847 - accuracy: 0.8916
Epoch 8/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1631 - precision: 0.9036 - recall: 0.8895 - accuracy: 0.8952
Epoch 9/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1691 - precision: 0.9029 - recall: 0.8878 - accuracy: 0.8963
Epoch 10/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1608 - precision: 0.9048 - recall: 0.8900 - accuracy: 0.8971
Epoch 11/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1642 - precision: 0.9030 - recall: 0.8886 - accuracy: 0.8957
Epoch 12/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1637 - precision: 0.9058 - recall: 0.8886 - accuracy: 0.8973
Epoch 13/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1761 - precision: 0.8990 - recall: 0.8814 - accuracy: 0.8915
Epoch 14/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1586 - precision: 0.9062 - recall: 0.8933 - accuracy: 0.9007
Epoch 15/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1674 - precision: 0.9039 - recall: 0.8890 - accuracy: 0.8954
Epoch 16/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1598 - precision: 0.9025 - recall: 0.8888 - accuracy: 0.8952
Epoch 17/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1672 - precision: 0.9022 - recall: 0.8867 - accuracy: 0.8953
Epoch 18/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1622 - precision: 0.9038 - recall: 0.8912 - accuracy: 0.8971
Epoch 19/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1646 - precision: 0.9078 - recall: 0.8954 - accuracy: 0.9010
Epoch 20/20
732/732 [==============================] - 3s 4ms/step - loss: 0.1685 - precision: 0.8989 - recall: 0.8831 - accuracy: 0.8915
```

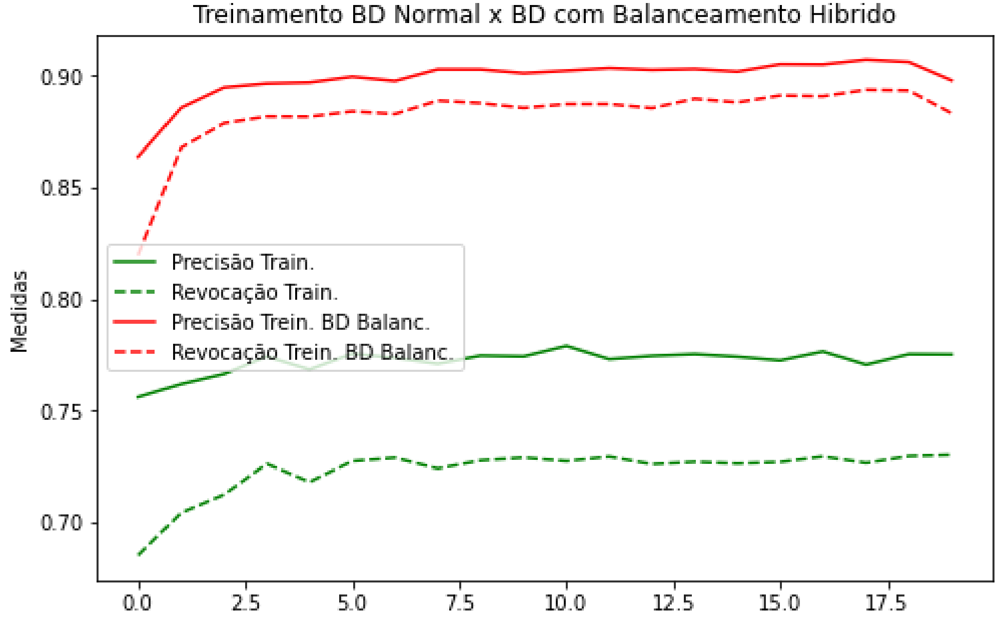
```python
hist_bdBalanc_t = model_bdBal.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=0)

score1_TrBal = model_bdBal.evaluate(x_train, y_train, verbose = 0)
score1_TeBal = model_bdBal.evaluate(x_test, y_test, verbose = 0)
```

```
score1_teBal = model_bdBal.evaluate(x_test, y_test, verbose = 0)
```

Resulta da RNN considerando a base de dados normal *versus* a RNN com base de dados com o tratamento do Balanceamento SMOTEENN (hibrido)

```
##-- Gráfico da precisão e revocação no treinamento S e teste T
plt.figure(figsize=(8,5))
plt.plot(hist_bdnormal.history['precision'], 'g')
plt.plot(hist_bdnormal.history['recall'], 'g--')
plt.plot(hist_bdBal.history['precision'], 'r')
plt.plot(hist_bdBal.history['recall'], 'r--')
plt.ylabel('Medidas' )
plt.legend(["Precisão Train.", "Revocação Train.", "Precisão Trein. BD Balanc.", "Revocação Trein. BD Balanc."], loc="best")
plt.title('Treinamento BD Normal x BD com Balanceamento Hibrido')
```

```
Text(0.5, 1.0, 'Treinamento BD Normal x BD com Balanceamento Hibrido')
```



>> Houve ganho significativo com o tramento da base com balanceamento Smoteenn (hibrido) com a utilização da rede neural.

```
print("Acurácia treinamento - BD Normal: %.4f" % (score1_Tr[1]))
print("Acurácia teste - BD Normal: %.4f" % (score1_Te[1]))

print("Acurácia treinamento - BD Balanc.: %.4f" % (score1_TrBal[1]))
print("Acurácia teste - BD Balanc.: %.4f" % (score1_TeBal[1]))
```

```
Acurácia treinamento - BD Normal: 0.7647
Acurácia teste - BD Normal: 0.7634
Acurácia treinamento - BD Balanc.: 0.8978
Acurácia teste - BD Balanc.: 0.9956
```

0s conclusão: 16:33