

AVALIACAO FINAL - GUILHERME LOURENCO

November 27, 2020

0.1 MBA em Ciência de Dados

1 Redes Neurais e Arquiteturas Profundas

1.1 Avaliação Final

Moacir Antonelli Ponti

CeMEAI - ICMC/USP São Carlos

Nessa avaliação será utilizado o dataset `creditcard.csv` que contém 31 colunas. Esse problema é o de detectar fraude em transações em cartões de crédito. Vamos assumir um cenário com alta disponibilidade de exemplos não rotulados, e baixa de exemplos rotulados. Para tal, pré-treinaremos camadas de uma rede neural com dados não anotados, a qual posteriormente será usada para compor um modelo inicial de classificação.

Você deverá criar um notebook (`ipynb`) com a solução. O Notebook deverá conter as saídas para visualização dos resultados. Fazer upload no Moodle de 2 arquivos:

1. Notebook com o código fonte (`ipynb`)
2. Versão em PDF com todos os resultados

Conforme código abaixo, use como características de entrada as colunas de posição 1 até 28 (marcadas no arquivo como `V1 - V28`), e como classe a última coluna (Class). Não utilize a coluna `Amount`.

As tarefas a realizar são as seguintes:

1. **Separe** os dados em:
 - conjunto S = 2,5% dos dados iniciais como treinamento com rótulo (assumiremos que temos rótulos apenas para esses 2,5%, ou 7120 exemplos), no formato par (x,y)
 - conjunto U = 50% dos dados iniciais como treinamento não anotado (note que S está contido em U),
 - conjunto T = o restante dos 50% para teste, no formato par (x,y) .
2. **Modelo A:** denoising overcomplete autoencoder para pré-treinamento baseado em auto-supervisão
 - Arquitetura com as seguintes camadas:
 - entrada com 28 valores
 - normalização em batch
 - densa 32 neurônios, relu
 - densa 32 neurônios, relu
 - dropout 0.2

- normalização em batch
 - densa 28 neurônios, relu (camada de código/bottleneck)
 - densa 32 neurônios, relu
 - densa 32 neurônios, relu
 - densa 28 neurônios, tanh
 - Inserção de ruído aleatório uniforme ponderado a 0.2 (insira ruído nos dados de treinamento fornecidos por entrada, mas mantenha a comparação com a saída sem ruído, como num denoising autoencoder)
 - Taxa de aprendizado inicial de 0.003 e com decaimento a partir da época 5, exponencial a -0.2
 - Treinar com perda MSE por 20 épocas com batch size 16 utilizando o conjunto **U**
- 3. Análise de projeção das características:** visualize um scatterplot com os 2 principais componentes obtidos do PCA com as classes dos exemplos atribuídas com cores ou marcadores diferentes:
- scatterplot com projeção PCA do conjunto de S original
 - scatterplot com projeção PCA do conjunto S após processado pelo “encoder”, ou seja resultado da saída da camada de código
- 4. Modelo B:** rede neural profunda densa, utilizando como base o encoder do modelo A, e inserindo uma nova camada densa de classificação com ativação sigmóide.
- Taxa de aprendizado inicial de 0.001 e com decaimento em todas as épocas exponencial a -0.3
 - Uso de pesos para as classes: 0.1 para classe 0 (majoritária), e 0.9 para a classe 1 (minoritária)
 - Treinar com perda MSE por 8 épocas com batch size 16
 - Compute como métricas, além da perda, precisão e revocação (precision / recall)
- 5. Avalie a rede neural de classificação:**
- Exiba o gráfico da precisão e revocação no treinamento calculada ao longo das épocas
 - Exiba precisão e revocação calculada no treinamento S e teste T
 - Exiba um scatterplot do conjunto S obtendo sua representação do código da rede de classificação (saída da camada com 28 exemplos)
- 6. Bônus:** (+1 ponto extra) compare a solução com duas outras possibilidades que não envolvam uso do conjunto **U** não rotulado
- Rede neural profunda com a mesma arquitetura e estratégias usadas no modelo B, mas sem usar os pesos pré-treinados, inicializando e treinando com os dados em S por 15 épocas. Avalie precisão e revocação no treinamento S e teste T.
 - Classificador SVM treinado nos dados originais S. Avalie precisão e revocação no treinamento S e teste T.
 - Classificador SVM treinado nos dados S obtendo sua representação do código da rede de classificação (modelo B). Avalie precisão e revocação no treinamento S e teste T.

```
[1]: 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

seed(1)
set_seed(2)
```

```
[ ]: import pandas as pd
df = pd.read_csv("creditcard.csv")
```

1.1.1 Parte 1: separar dados

Conjunto df

- Tamanho: 284.807 Registros
- Variáveis: 30 Variáveis (28 utilizadas)
- Classes: 2 Classes

```
[3]: X = df.filter(regex='[v\d+]').copy()
y = df[['Class']].copy()
size = len(df)

print(f'Conjunto df:\n{X.shape[1]} Variáveis | {y.unique()[0]} Classes |_
→{size} Registros')
```

Conjunto df:
28 Variáveis | 2 Classes | 284807 Registros

Conjunto S

- Tamanho: 2,5% iniciais de df
- Variáveis: 28 Variáveis
- Classes: 2 Classes

```
[4]: S = [X.iloc[:int(size*0.025)].copy(), y.iloc[:int(size*0.025)].copy()]

print(f'Conjunto S:\n{S[0].shape[1]} Variáveis | {S[1].unique()[0]} Classes |_
→{S[0].shape[0]} Registros')
```

Conjunto S:
28 Variáveis | 2 Classes | 7120 Registros

Conjunto U

- Tamanho: 50% iniciais de df
- Variáveis: 28 Variáveis
- Classes: 2 Classes

```
[5]: U = [X.iloc[:int(size/2)].copy(), y.iloc[:int(size/2)].copy()]

print(f'Conjunto U:\n{U[0].shape[1]} Variáveis | {U[1].unique()[0]} Classes |_
→{U[0].shape[0]} Registros')
```

Conjunto U:
28 Variáveis | 2 Classes | 142403 Registros

Conjunto T

- Tamanho: 50% finais de df
- Variáveis: 28 Variáveis
- Classes: 2 Classes

```
[6]: T = [X.iloc[int(size/2):].copy(), y.iloc[int(size/2):].copy()]

print(f'Conjunto T:\n{T[0].shape[1]} Variáveis | {T[1].nunique()[0]} Classes |'
      f'{T[0].shape[0]} Registros')
```

Conjunto T:
28 Variáveis | 2 Classes | 142404 Registros

1.1.2 Parte 2: Modelo A

```
[7]: input_data = keras.layers.Input(shape=28)
network = keras.layers.BatchNormalization()(input_data)
# densa 32 neurônios, relu
network = keras.layers.Dense(32, activation='relu')(network)
# densa 32 neurônios, relu
network = keras.layers.Dense(32, activation='relu')(network)
# dropout 0.2
network = keras.layers.Dropout(0.2)(network)
# normalização em batch
network = keras.layers.BatchNormalization()(network)
# densa 28 neurônios, relu (camada de código/bottleneck)
network = keras.layers.Dense(28, activation='relu', name='code')(network)
# densa 32 neurônios, relu
network = keras.layers.Dense(32, activation='relu')(network)
# densa 32 neurônios, relu
network = keras.layers.Dense(32, activation='relu')(network)
# densa 28 neurônios, tanh
output = keras.layers.Dense(28, activation = 'tanh')(network)

modelA = keras.models.Model(input_data, output)
```

```
[8]: modelA.summary()
```

```
Model: "functional_1"
-----
Layer (type)          Output Shape       Param #
=====
input_1 (InputLayer)   [(None, 28)]        0
-----
batch_normalization (BatchNo (None, 28)    112
```

```

-----  

dense (Dense)           (None, 32)      928  

-----  

dense_1 (Dense)          (None, 32)      1056  

-----  

dropout (Dropout)        (None, 32)      0  

-----  

batch_normalization_1 (Batch (None, 32)    128  

-----  

code (Dense)             (None, 28)      924  

-----  

dense_2 (Dense)          (None, 32)      928  

-----  

dense_3 (Dense)          (None, 32)      1056  

-----  

dense_4 (Dense)          (None, 28)      924  

=====  

Total params: 6,056  

Trainable params: 5,936  

Non-trainable params: 120
-----
```

[9]:

```
batch_size = 16
epochs = 20
noise_factor = 0.2
```

[10]:

```
x_train = np.array(U[0])

x_train_noised = x_train + noise_factor * np.random.normal(0, 1, x_train.shape)
```

[11]:

```
def scheduler_modelA(epoch, lr):
    if epoch < 5:
        return lr
    else:
        return np.round(lr * tf.math.exp(-0.2), 4)
```

[12]:

```
callback = keras.callbacks.LearningRateScheduler(scheduler_modelA)
```

[13]:

```
modelA.compile(
    loss = 'mse',
    optimizer = keras.optimizers.Adam(lr = 0.003)
)
```

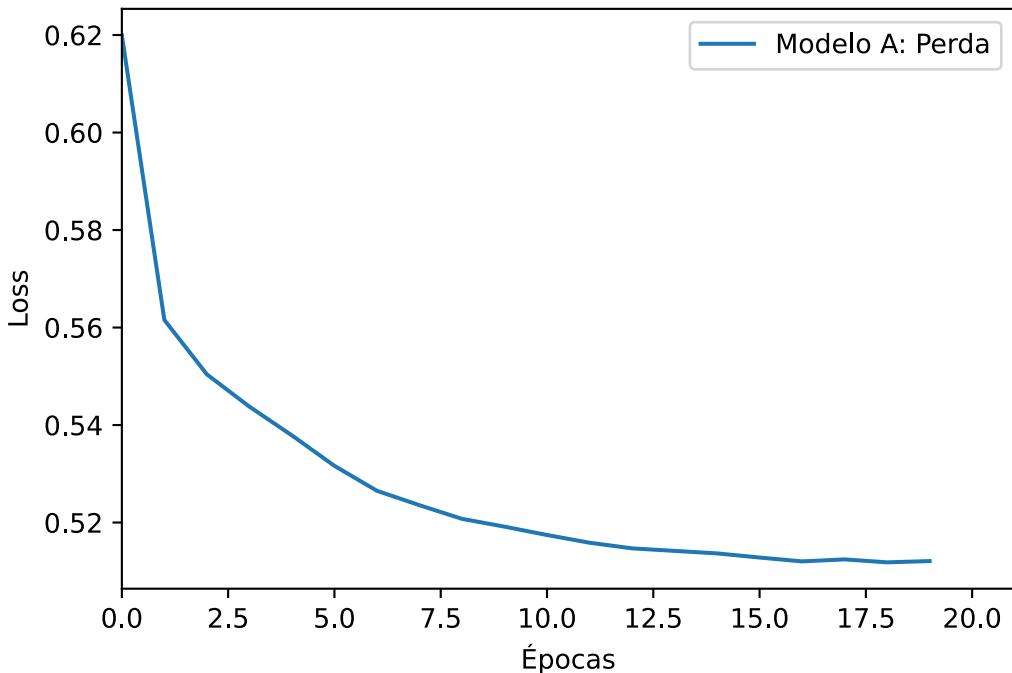
[14]:

```
histA = modelA.fit(
    x = x_train_noised,
    y = x_train,
    batch_size = batch_size,
```

```
    epochs = epochs,  
    callbacks = [callback],  
    verbose = 1  
)
```

```
Epoch 1/20  
8901/8901 [=====] - 36s 4ms/step - loss: 0.6200  
Epoch 2/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5616  
Epoch 3/20  
8901/8901 [=====] - 36s 4ms/step - loss: 0.5504  
Epoch 4/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5438  
Epoch 5/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5379  
Epoch 6/20  
8901/8901 [=====] - 37s 4ms/step - loss: 0.5316  
Epoch 7/20  
8901/8901 [=====] - 36s 4ms/step - loss: 0.5265  
Epoch 8/20  
8901/8901 [=====] - 36s 4ms/step - loss: 0.5236  
Epoch 9/20  
8901/8901 [=====] - 36s 4ms/step - loss: 0.5208  
Epoch 10/20  
8901/8901 [=====] - 36s 4ms/step - loss: 0.5192  
Epoch 11/20  
8901/8901 [=====] - 37s 4ms/step - loss: 0.5174  
Epoch 12/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5159  
Epoch 13/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5147  
Epoch 14/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5142  
Epoch 15/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5137  
Epoch 16/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5128  
Epoch 17/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5120  
Epoch 18/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5124  
Epoch 19/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5118  
Epoch 20/20  
8901/8901 [=====] - 38s 4ms/step - loss: 0.5121
```

```
[15]: plt.plot(histA.history['loss'])
plt.xlim(0, 21)
plt.ylabel('Loss')
plt.xlabel('Épocas')
plt.legend(['Modelo A: Perda'])
plt.show();
```



1.1.3 Parte 3: Análise da projeção das características

```
[16]: from sklearn.decomposition import PCA
```

```
[17]: x_train = np.array(S[0])
y_train = np.array(S[1])
```

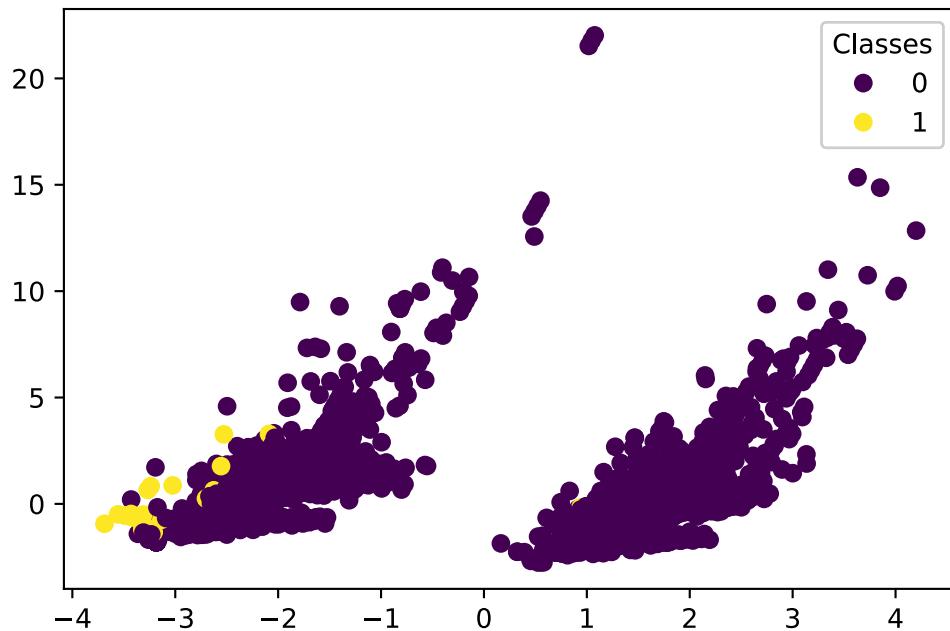
```
[20]: pca = PCA(n_components = 2)
pca.fit(x_train)
pca_train = pca.transform(x_train)
```

```
[21]: fig, ax = plt.subplots()
scatter = ax.scatter(
    pca_train[:,0]
    , pca_train[:,1]
    , c = y_train[:]
    , cmap='viridis'
```

```

        )
legend = ax.legend(
    *scatter.legend_elements()
    , loc='best'
    , title='Classes'
)
ax.add_artist(legend);

```



```
[22]: modelCode = keras.models.Model(
    inputs = modelA.input
    ,   outputs = modelA.get_layer('code').output
)
```

```
[23]: codeTrain = np.asarray(modelCode.predict(x_train))
```

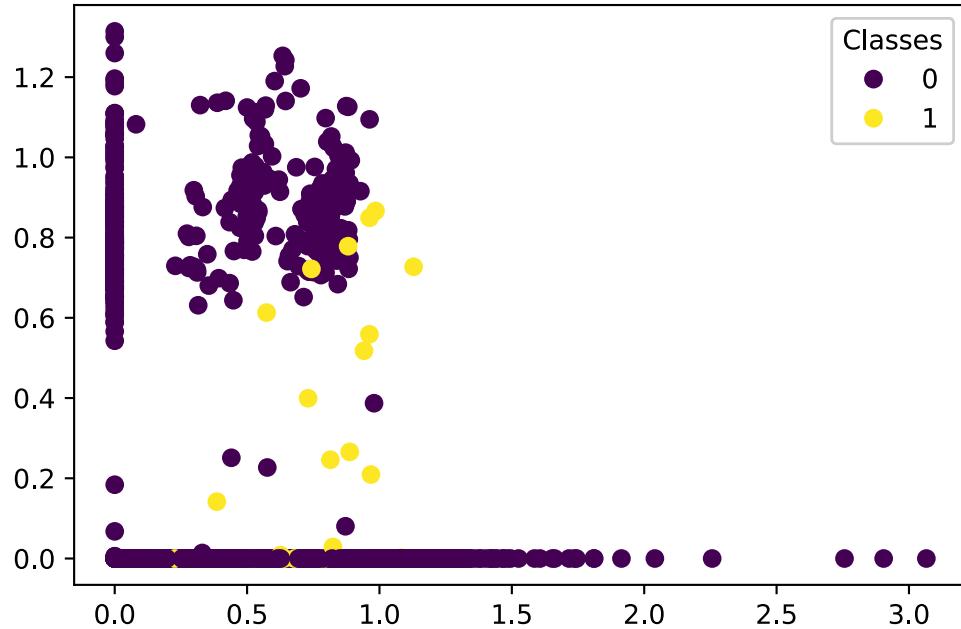
```
[24]: pca = PCA(n_components = 2)
pca.fit(codeTrain)
pca_train = pca.transform(codeTrain)
```

```
[25]: fig, ax = plt.subplots()
scatter = ax.scatter(
    codeTrain[:,0]
    ,   codeTrain[:,1]
    ,   c = y_train[:]
    ,   cmap='viridis'
```

```

        )
legend = ax.legend(
    *scatter.legend_elements(),
    loc='best',
    title='Classes'
)
ax.add_artist(legend);

```



1.1.4 Parte 4: Modelo B

[26]:

```
encoder = keras.models.Model(
    inputs = modelA.input
    ,   outputs = modelA.get_layer(name = 'code').output
)
```

[27]:

```
update_model = keras.models.Sequential()
update_model.add(
    keras.layers.Dense(
        1,
        activation = 'sigmoid',
        name = 'pred',
        input_shape = encoder.output_shape[1:]
    )
)
```

```
modelB = keras.models.Model(  
    inputs = encoder.input,  
    outputs = update_model(encoder.output)  
)
```

[28]: modelB.summary()

```
Model: "functional_7"  
-----  
Layer (type)          Output Shape         Param #  
=====-----  
input_1 (InputLayer)   [(None, 28)]        0  
-----  
batch_normalization (BatchNo (None, 28)      112  
-----  
dense (Dense)          (None, 32)           928  
-----  
dense_1 (Dense)        (None, 32)           1056  
-----  
dropout (Dropout)      (None, 32)           0  
-----  
batch_normalization_1 (Batch (None, 32)       128  
-----  
code (Dense)            (None, 28)           924  
-----  
sequential (Sequential) (None, 1)            29  
=====-----  
Total params: 3,177  
Trainable params: 3,057  
Non-trainable params: 120  
-----
```

[29]: batch_size = 16
epochs = 8
class_weight = {0: 0.1, 1: 0.9}

[30]: x_train = np.array(U[0])
y_train = np.array(U[1])

[31]: def scheduler_modelB(epoch, lr):
 return np.round(lr * tf.math.exp(-0.3), 4)

[32]: callback = keras.callbacks.LearningRateScheduler(scheduler_modelB)

[33]: modelB.compile(
 loss = 'mse',
 optimizer = keras.optimizers.Adam(lr = 0.001),

```
    metrics = [
        tf.keras.metrics.Precision(),
        tf.keras.metrics.Recall()
    ]
)
```

```
[34]: histB = modelB.fit(
    x = x_train,
    y = y_train,
    batch_size = batch_size,
    epochs = epochs,
    callbacks = [ callback ],
    class_weight = class_weight,
    verbose = 1
)
```

```
Epoch 1/8
8901/8901 [=====] - 56s 6ms/step - loss: 7.8191e-04 -
precision: 0.6736 - recall: 0.6059
Epoch 2/8
8901/8901 [=====] - 58s 6ms/step - loss: 3.7629e-04 -
precision: 0.7726 - recall: 0.7955
Epoch 3/8
8901/8901 [=====] - 56s 6ms/step - loss: 3.5011e-04 -
precision: 0.7711 - recall: 0.8141
Epoch 4/8
8901/8901 [=====] - 56s 6ms/step - loss: 3.4591e-04 -
precision: 0.7794 - recall: 0.8141
Epoch 5/8
8901/8901 [=====] - 57s 6ms/step - loss: 3.3079e-04 -
precision: 0.7809 - recall: 0.8216
Epoch 6/8
8901/8901 [=====] - 56s 6ms/step - loss: 3.4385e-04 -
precision: 0.7766 - recall: 0.8141
Epoch 7/8
8901/8901 [=====] - 56s 6ms/step - loss: 3.3039e-04 -
precision: 0.7957 - recall: 0.8253
Epoch 8/8
8901/8901 [=====] - 56s 6ms/step - loss: 3.1574e-04 -
precision: 0.7805 - recall: 0.8327
```

1.1.5 Parte 5: Avaliação da rede neural de classificação

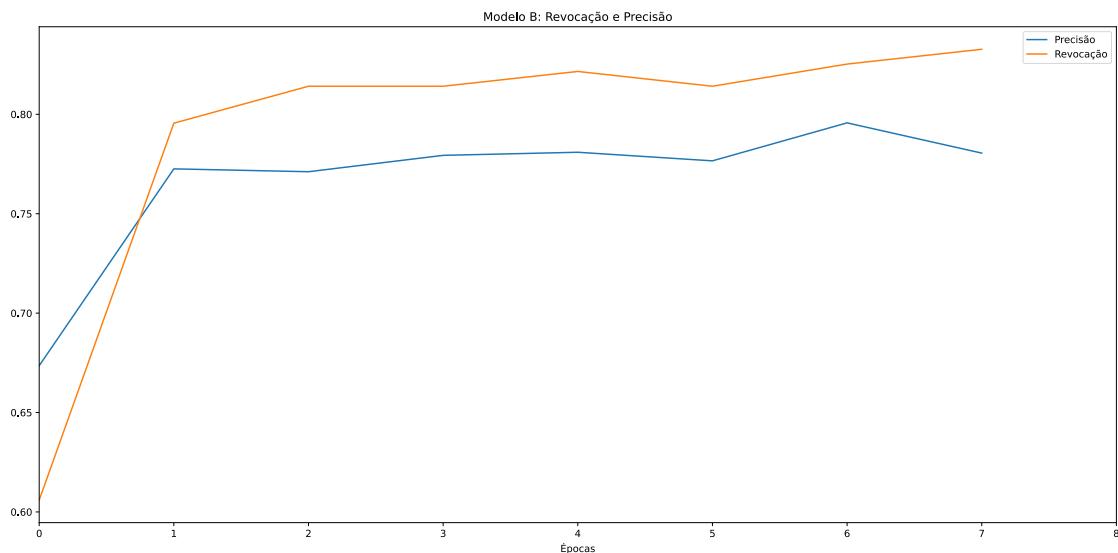
```
[35]: fig, axes = plt.subplots(figsize = (16,8))

plt.plot(histB.history['precision'], label = 'precision')
plt.plot(histB.history['recall'], label = 'recall')
```

```

plt.title('Modelo B: Revocação e Precisão')
plt.xlabel('Épocas')
plt.xlim(0, 8)
plt.legend(['Precisão', 'Revocação'], loc = 'best')
fig.tight_layout()
plt.show();

```



```
[36]: score_S = modelB.evaluate(S[0], S[1], verbose=0)
print(f'Conjunto S:\nPrecisão: {np.round(score_S[1], 4)}\nRevocação: {np.
    round(score_S[2], 4)}')
```

Conjunto S:
Precisão: 0.92
Revocação: 0.92

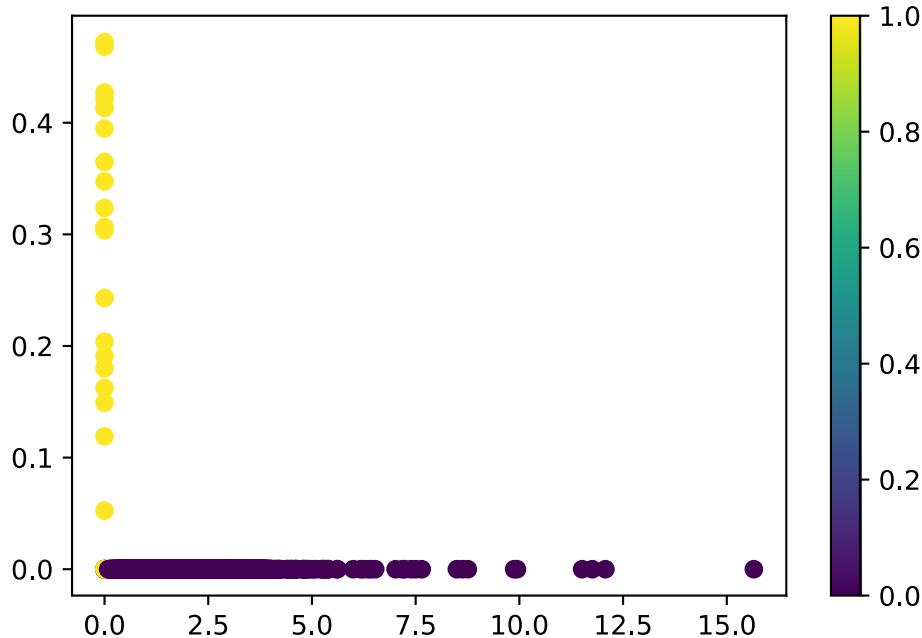
```
[37]: score_T = modelB.evaluate(T[0], T[1], verbose=0)
print(f'Conjunto T:\nPrecisão: {np.round(score_T[1], 4)}\nRevocação: {np.
    round(score_T[2], 4)}')
```

Conjunto T:
Precisão: 0.8469
Revocação: 0.7444

```
[38]: codeModel = keras.models.Model(
    inputs = modelB.input,
    outputs = modelB.get_layer('code').output
)
```

```
[39]: trainCode = np.asarray(codeModel.predict(S[0]))
```

```
[40]: plt.scatter(
    trainCode[:,0], trainCode[:,1], c = S[1][:].values, cmap='viridis'
)
plt.colorbar();
```



1.1.6 Bônus: Tentando outros métodos para treinar com os poucos dados rotulados

Tentativa 1: DNN com a mesma arquitetura usada, mas sem pré-treinamento

```
[41]: encoderBonus = keras.models.Model(
    inputs = modelA.input,
    outputs = modelA.get_layer(name = 'code').output
)
```

```
[42]: update_model = keras.models.Sequential()
update_model.add(
    keras.layers.Dense(
        1,
        activation = 'sigmoid',
        name = 'pred',
        input_shape = encoderBonus.output_shape[1:]
    )
)
```

```
modelDNN = keras.models.Model(  
    inputs = encoderBonus.input,  
    outputs = update_model(encoderBonus.output)  
)
```

[43]: modelDNN.summary()

```
Model: "functional_13"  
-----  
Layer (type)          Output Shape         Param #  
=====-----  
input_1 (InputLayer)   [(None, 28)]        0  
-----  
batch_normalization (BatchNo (None, 28)      112  
-----  
dense (Dense)          (None, 32)           928  
-----  
dense_1 (Dense)         (None, 32)           1056  
-----  
dropout (Dropout)       (None, 32)           0  
-----  
batch_normalization_1 (Batch (None, 32)       128  
-----  
code (Dense)            (None, 28)           924  
-----  
sequential_1 (Sequential) (None, 1)            29  
=====-----  
Total params: 3,177  
Trainable params: 3,057  
Non-trainable params: 120
```

[44]: batch_size = 16
epochs = 15

[45]: x_train = np.array(S[0])
y_train = np.array(S[1])

[46]: def schedulerDNN(epoch, lr):
 return np.round(lr * tf.math.exp(-0.3), 4)

[47]: callback = keras.callbacks.LearningRateScheduler(schedulerDNN)

[48]: modelDNN.compile(
 loss = 'mse',
 optimizer = keras.optimizers.Adam(lr = 0.001),
 metrics = [

```
        tf.keras.metrics.Precision(),
        tf.keras.metrics.Recall()
    ]
)
```

```
[49]: histDNN = modelDNN.fit(
    x = x_train,
    y = y_train,
    batch_size = batch_size,
    epochs = epochs,
    callbacks = [ callback],
    verbose = 1
)
```

```
Epoch 1/15
445/445 [=====] - 2s 5ms/step - loss: 0.0520 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 2/15
445/445 [=====] - 2s 5ms/step - loss: 0.0042 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 3/15
445/445 [=====] - 2s 5ms/step - loss: 0.0037 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 4/15
445/445 [=====] - 2s 4ms/step - loss: 0.0036 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 5/15
445/445 [=====] - 2s 4ms/step - loss: 0.0035 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 6/15
445/445 [=====] - 2s 4ms/step - loss: 0.0035 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 7/15
445/445 [=====] - 2s 5ms/step - loss: 0.0035 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 8/15
445/445 [=====] - 2s 5ms/step - loss: 0.0034 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 9/15
445/445 [=====] - 2s 5ms/step - loss: 0.0034 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 10/15
445/445 [=====] - 2s 4ms/step - loss: 0.0034 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 11/15
445/445 [=====] - 2s 4ms/step - loss: 0.0033 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
```

```
Epoch 12/15
445/445 [=====] - 2s 5ms/step - loss: 0.0032 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 13/15
445/445 [=====] - 2s 5ms/step - loss: 0.0030 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 14/15
445/445 [=====] - 2s 4ms/step - loss: 0.0026 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
Epoch 15/15
445/445 [=====] - 2s 4ms/step - loss: 0.0023 -
precision_1: 0.0000e+00 - recall_1: 0.0000e+00
```

```
[50]: score_S = modelDNN.evaluate(S[0], S[1], verbose=0)
print(f'Conjunto S:\nPrecisão: {np.round(score_S[1], 4)}\nRevocação: {np.
    round(score_S[2], 4)}')
```

Conjunto S:
Precisão: 1.0
Revocação: 0.16

```
[51]: score_T = modelDNN.evaluate(T[0], T[1], verbose=0)
print(f'Conjunto T:\nPrecisão: {np.round(score_T[1], 4)}\nRevocação: {np.
    round(score_T[2], 4)}')
```

Conjunto T:
Precisão: 1.0
Revocação: 0.0045

Tentativa 2: SVM nos dados originais

```
[52]: x_train = np.array(S[0])
y_train = np.array(S[1])
```

```
[53]: x_test = np.array(T[0])
y_test = np.array(T[1])
```

```
[54]: from sklearn import svm
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

clf = svm.SVC(C = 10, random_state = 42)

clf.fit(x_train, y_train)
```

```
[54]: SVC(C=10, random_state=42)
```

```
[55]: y_pred = clf.predict(x_train)

print(f'Conjunto S:\nPrecisão: {np.round(precision_score(y_train, y_pred), 4)}\nRevocação: {np.round(recall_score(y_train, y_pred), 4)}')
```

Conjunto S:
Precisão: 1.0
Revocação: 1.0

```
[56]: y_pred = clf.predict(x_test)

print(f'Conjunto T:\nPrecisão: {np.round(precision_score(y_test, y_pred), 4)}\nRevocação: {np.round(recall_score(y_test, y_pred), 4)}')
```

Conjunto T:
Precisão: 0.6538
Revocação: 0.1525

```
[57]: modelSVM = keras.models.Model(
    inputs = modelB.input,
    outputs = modelB.get_layer('code').output
)
```

```
[58]: codeSVM_train = np.array(modelSVM.predict(np.array(S[0])))
y_train = np.array(S[1])
```

```
[59]: codeSVM_test = np.array(modelSVM.predict(np.array(T[0])))
y_test = np.array(T[1])
```

```
[60]: from sklearn import svm
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

clf = svm.SVC(C = 10, random_state = 1)

clf.fit(codeSVM_train, y_train)
```

```
[60]: SVC(C=10, random_state=1)
```

```
[61]: y_pred = clf.predict(codeSVM_train)

print(f'Conjunto S:\nPrecisão: {np.round(precision_score(y_train, y_pred), 4)}\nRevocação: {np.round(recall_score(y_train, y_pred), 4)}')
```

Conjunto S:
Precisão: 1.0
Revocação: 0.92

```
[62]: y_pred = clf.predict(codeSVM_test)

print(f'Conjunto T:\nPrecisão: {np.round(precision_score(y_test, y_pred), 4)}\nRevocação: {np.round(recall_score(y_test, y_pred), 4)}')
```

Conjunto T:
Precisão: 0.7917
Revocação: 0.2556

```
[ ]:
```